IBM RT Advanced Interactive Executive Operating System Version 2.2.1

Interface Program for use with TCP/IP

Communication Family

IBM

SC23-2005-1

IBM RT Advanced Interactive Executive Operating System Version 2.2.1

Interface Program for use with TCP/IP

Communication Family

IBM

Second Edition (September 1988)

Portions of the code and documentation described in this book were developed at the Electrical Engineering and Computer Sciences Department at the Berkeley Campus of the University of California under the auspices of the Regents of the University of California.

This edition applies to Version 2.2.1 of IBM RT Interface Program for use with TCP/IP, and to all subsequent releases until otherwise indicated in new editions or technical newsletters. Changes are made periodically to the information herein; these changes will be reported in technical newsletters or in new editions of this publication.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

International Business Machines Corporation provides this manual "as is," without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this manual at any time.

Products are not stocked at the address given below. Requests for copies of this product and for technical information about the system should be made to your IBM representative, your IBM Authorized Dealer, or your IBM Authorized Remarketer.

A reader's comment form is provided at the back of this publication. If the form has been removed, address comments to IBM Corporation, Department 997, 11400 Burnet Road, Austin, Texas 78758-3493. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Permission to use, copy, modify, and distribute this software only for the purposes and only in the manner set forth in the appertaining agreement is hereby granted, provided that this copyright and permission notice appear on all copies and supporting documentation. Further, the name of M.I.T. is not to be used in any advertising or publicity pertaining to distribution of the software without specific prior permission.

IBM is a registered trademark of International Business Machines Corporation.
RT is a registered trademark of International Business Machines Corporation.

© Copyright International Business Machines Corporation 1985, 1987, 1988
© Copyright Massachusetts Institute of Technology 1984
© Copyright Paul G. Milazzo 1985, all rights reserved

About This Book

This book provides information about the IBM RT® Interface Program for use with TCP/IP (Transmission Control Protocol/Internet Protocol). With the Interface Program for use with TCP/IP, you can communicate with:

- Other IBM RT systems that also have the Interface Program installed
- Other host systems that support TCP/IP.

Note: In this book, the term ***Interface Program*** refers to the IBM RT Interface Program for use with TCP/IP.

Who Should Use This Book

This book is intended for Interface Program users, system managers, and communications programmers. The major tasks covered in this book are:

- Transferring data between an IBM RT system and another host computer
- Using another IBM RT system or host, or its facilities, remotely
- Managing networks.

For programmers, this book also describes the Application Programming Interfaces (APIs) for the Interface Program, including the library routines.

Readers require a thorough understanding of the IBM RT functions. In addition, programmers will find knowledge of the Advanced Interactive Executive (AIX™) Operating System helpful.

RT is a registered trademark of International Business Machines Corporation.

AIX is a trademark of International Business Machines Corporation.

Before You Begin

Before you can use the Interface Program, you must have installed the following software and hardware on each IBM RT system that is to be on the network:

Software

- IBM RT AIX Operating System licensed program, Version 2.2.1
- Depending on the hardware installed in your system, one or more of the following VRM Device Drivers:
  - IBM RT VRM Baseband Adapter Device Driver
  - IBM RT VRM Token-Ring Device Driver
  - A customer-supplied VRM X.25 Adapter Device Driver
  - A customer-supplied VRM 802.3 Adapter Device Driver.

Hardware

- One or more of the following adapters:
  - IBM RT Baseband Adapter for use with Ethernet[1]
  - IBM Token-Ring Network RT Adapter
  - A customer-supplied X.25 Adapter
  - A customer-supplied 802.3 Adapter
  - IBM RT 6150 Native Serial Ports
  - IBM Personal Computer AT Serial/Parallel Adapter
  - IBM RT Buffered 4-Port Asynchronous RS-232C Adapter
  - IBM RT Buffered 4-Port Asynchronous RS-422A Adapter
  - IBM RT 8-Port RS-232C Asychronous Adapter
  - IBM RT 8-Port RS-422A Asychronous Adapter
  - IBM RT 8-Port MIL-STD 188 Adapter
- Cables and connectors.

For further information on configuring these adapters into your system, see the section on communications hardware adapters in *IBM RT Planning Guide.*

Note: In this book, the term ***IBM Asynchronous Adapter*** refers to any of the asynchronous adapters in the preceding list.

Refer to the following publications for information about installing the software and hardware:

- *IBM RT Installing and Customizing the AIX Operating System*

---

1 Ethernet is a trademark of Xerox, Inc.

- *IBM RT Options Installation*
- *IBM RT Planning Guide*
- *IBM RT User Setup Guide.*

How to Use This Book

After you install the necessary software and hardware, you should become familiar with the information in Chapter 1, "General Information" before you refer to Appendix A, "Customizing the Program" for information about creating an Interface Program network to meet your requirements.

Once you have installed and customized the Interface Program, you should become familiar with the information in Chapter 2, "User Commands." For details on managing the network, refer to Chapter 3, "Server Commands" and Chapter 4, "File Formats." With the information in these two chapters, you should be able to use the Interface Program commands to transfer data, log in remotely, and manage the network.

If you want to use the Application Programming Interfaces (APIs) to the Internet protocols, you also should become familiar with the information on the sockets interface described in *AIX Operating System Technical Reference.*

Organization

This book is divided into the following chapters:

- Chapter 1, "General Information" provides an overview of the Interface Program.
- Chapter 2, "User Commands" describes Interface Program user commands.
- Chapter 3, "Server Commands" describes the Interface Program server commands, or daemons.
- Chapter 4, "File Formats" describes the formats of files used to customize the Interface Program.

The following appendixes contain supplemental information:

- Appendix A, "Customizing the Program" explains how to adapt the Interface Program to your requirements after you install it.
- Appendix B, "Examples" provides programming examples for the sockets interface to the Internet protocols.
- Appendix C, "Protocol Library Routines" describes the Interface Program **tcpm**, **tcp**, and **udp** protocol libraries.

A Reader's Comment Form and Book Evaluation Form are provided at the back of this book. Use the Reader's Comment Form at any time to give IBM information that may

improve the book. After you become familiar with the book, use the Book Evaluation Form to give IBM specific feedback about the book.

Typography

This book uses type style to distinguish among kinds of information. General information is printed in the standard type style (the style used for this sentence). The following type styles indicate other types of information:

New terms
The first occurrence of each new term is printed in this style.

System parts
Names of commands, files, and other parts of the system are printed in this style.

Variable information
Names for information you must provide are printed in this style.

`Information you are to type`
To run the examples in this book, enter the information printed in this style.

Related Publications

The following documents in the IBM RT series contain additional information that may prove helpful in understanding and using the Interface Program:

- The following volumes in the *IBM RT AIX Operating System Technical Reference*:
  - *System Calls and Subroutines*
  - *VRM Device Support.*

See *IBM RT Bibliography and Master Index* for order numbers of IBM RT publications and diskettes.

For general information about TCP/IP, the following publications are recommended. These publications are distributed by the Network Information Center on behalf of the Defense Communications Agency and Defense Advanced Research Projects Agency (DARPA). The mailing address is:

Network Information Center
SRI International
Menlo Park, CA 92025

- *Broadcasting Internet Datagrams*, RFC 919, J. Mogul
- *DARPA Internet Gateway*, RFC 823, R. Hinden, A. Sheltzer
- *DCN Local-Network Protocols*, RFC 891, D. L. Mills

- *Domain Administrators Guide*, RFC 1032, M. Stahl
- *Domain Administrators Operations Guide*, RFC 1033, M. Lottor
- *Domain Names - Concepts and Facilities*, RFC 1034, P. Mockapetris
- *Domain Names - Implementations and Specification*, RFC 1035, P. Mockapetris
- *An Ethernet Address Resolution Protocol*, RFC 826, D. Plummer
- *Exterior Gateway Protocol Formal Specification*, RFC 904, D. Mills
- *Exterior Gateway Protocol Implementation Schedule*, RFC 890, J. Postel
- *File Transfer Protocol*, RFC 959, J. Postel
- *Internet Control Message Protocol*, RFC 792, J. Postel
- *Internet Name Server Protocol*, IEN116, J. Postel
- *Internet Assigned Numbers*, RFC 1010, J. Reynolds, J. Postel
- *Internet Protocol*, RFC 791, J. Postel
- *Internet Standard Subnetting Procedure*, RFC 950, J. Mogul
- *Mail Routing and the Domain System*, RFC 974, C. Partridge
- *Name/Finger*, RFC 742, K. Harrenstien
- *Official Internet Protocols*, RFC 1011, J. Reynolds, J. Postel
- *Revised IP Security Option*, RFC 1038, M. St. Johns
- *A Standard for the Transmission of IP Datagrams over IEEE 802 Networks*, RFC 1042, J. Postel, J. Reynolds
- *Stub Exterior Gateway Protocol*, RFC 888, L. Seamonson, E. Rosen
- *Telnet Binary Transmission*, RFC 856, J. Postel, J. Reynolds
- *Telnet Option Specifications*, RFC 855, J. Postel, J. Reynolds
- *Telnet Protocol Specification*, RFC 854, J. Postel, J. Reynolds
- *Telnet Terminal Type Option*, RFC 930, M. Solomon, E. Wimmers
- *Time Protocol*, RFC 868, J. Postel, K. Harrenstien
- *Trailer Encapsulations*, RFC 893, S. Leffler, M. Karels
- *Transmission Control Protocol*, RFC 793, J. Postel
- *Trivial File Transfer Protocol*, RFC 783, K. R. Sollins
- *User Datagram Protocol*, RFC 768, J. Postel

For information about X.25 host interface specifications, consult the following publication, available from:

Defense Data Network Program Management Office
Defense Communications Agency
Code B610
Washington, D.C. 20305

- *Defense Data Network X.25 Host Interface Specifications*, December 1983, Defense Communications Agency. (Prepared by BBN Communications Corporation.)

For additional information about the IBM Token-Ring Network, you may want to order *IBM Token-Ring Architecture Reference* (Order Number 6165877) from your IBM representative.

Ordering Additional Copies of This Book

To order additional copies of this publication (without diskettes), use either of the following sources:

- To order from your IBM representative, use Order Number SBOF-1807.
- To order from your IBM dealer, use Part Number 27F4356.

A binder and *IBM RT Interface Program for use with TCP/IP* are included with the order. For information on ordering the binder or manual separately, contact your IBM representative or your IBM dealer.

Contents

Chapter 1. General Information

CONTENTS

About This Chapter

The IBM RT Interface Program for use with TCP/IP consists primarily of protocols and commands (application programs) that enable the RT system to communicate with:

- Other RT systems with the Interface Program installed
- Host systems that support TCP/IP.

This chapter provides an overview of the Interface Program protocols and commands, and also explains the other features and conventions of the program.

Overview

A Brief Look at the Interface Program for use with TCP/IP

The Interface Program provides facilities that make the RT system an Internet host that attaches to a network and communicates with other Internet hosts. The Interface Program for use with TCP/IP includes commands and facilities that allow you to:

- Transfer files between systems
- Log in to remote systems
- Run commands on remote systems
- Print files on remote systems
- Converse interactively with remote users
- Manage a network.

For those who want to develop programs using components of the Interface Program, the following Application Programming Interfaces are provided:

- Sockets interface to the Internet Protocol (IP), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP)
- Protocol library routines for Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), which are provided for compatibility with previous releases. New programs should use the sockets interface to the protocols.

Note: The **mail** user command, the Message Handling (MH) user commands, and the **sendmail** server command can use the Interface Program for sending and receiving mail between systems, and the Basic Network Utilities can use the Interface Program for sending and receiving files and commands between systems. For detailed information on these programs, refer to *IBM RT AIX Operating System Commands Reference*, *IBM RT Using the AIX Operating System*, and *IBM RT Managing the AIX Operating System*. Since these programs are not part of the Interface Program, they are not described in detail in this book.

The AIX Operating System contains sockets library routines, which provide the Interface Program with access to TCP/IP protocols. Refer to *AIX Operating System Technical Reference* for more information on the sockets library routines.

Internet Terms

Before continuing, you may find it useful to become familiar with the following Internet terms as they are used in this book.

client A computer or process that accesses the data, services, or resources of another computer or process on the network.

host A computer that is attached to an Internet network and can communicate with other Internet hosts. The ***local host*** for a particular user is the computer at which that user is working. A ***foreign host*** is any other host on the network. From the point of view of the communication network, hosts are the sources and destinations of packets. Any host can be a client, a server, or both. On an Internet network, a host is identified by its Internet name and address.

network The combination of two or more hosts and the connecting links between them. A ***physical network*** is the hardware that makes up the network. A ***logical network*** is the abstract organization overlaid on all or part of one or more physical networks. The Internet network is an example of a logical network. The Interface Program handles translation of logical network operations into physical network operations.

packet The block of control information and data for one transaction between a host and its network. Packets are the exchange medium used by processes to send and receive data through Internet networks. A packet is sent from a ***source*** to a ***destination***.

port A logical connecting point for a process. Data is transmitted between processes through ports (or ***sockets***). Each port provides queues for sending and receiving data. In an Interface Program network, each port has an Internet ***port number*** based on how it is being used. A particular port is identified with an Internet ***socket address***, which is the combination of an Internet host address and a port number.

process A program that is running. A process is the active element in a computer. Terminals, files, and other I/O devices communicate with each other through processes. Thus, network communication is ***interprocess communication*** (that is, communication between processes).

protocol A set of rules for handling communications at the physical or logical level. Protocols often use other protocols to provide services. For example, a ***connection-level protocol*** uses a ***transport-level protocol*** to transport packets that maintain a connection between two hosts.

server A computer or process that provides data, services, or resources that can be accessed by other computers or processes on the network.

Internet Environment

The AIX network protocols include the Internet Protocol (IP), higher-level Internet protocols, and other protocols that use the Internet address format. These protocols use services provided by the AIX Operating System on the RT system. The Interface Program provides facilities that make the RT system an Internet host that attaches to a network and communicates with other Internet hosts.

The Internet environment consists of hosts organized in logical networks that use ***packet-switched*** technology (that is, data is transferred in packets). These hosts are connected to physical networks that may or may not use packet-switched technology. Physical and logical networks, in turn, can be interconnected via gateway hosts. Packets can be ***routed*** between hosts that are on different networks through one or more gateway hosts.

Each Internet host is assigned at least one ***Internet host name*** that is used when entering commands. These hosts may or may not be assigned to a ***domain***. A domain is made up of a group of hosts or subdomains under the same administrative authority. Domains are given symbolic names. Host names must be unique within a domain. If domains are not used, host names must be unique within the local network.

Each Internet host is also assigned at least one unique ***Internet address***. This 32-bit address is used by the Internet Protocol (IP) and higher-level protocols. In the case of gateway hosts, more than one address may be assigned, and each interface to the Internet network is assigned its own unique address. Packets are routed between Internet hosts using the Internet addresses of the hosts involved in transporting the packets.

Internet addresses are made up of a ***network number*** and a ***local address***. A unique, official network number is assigned to each local network when it connects to other Internet networks. However, if a local network is not going to connect to other Internet networks, it can be assigned any network number that is convenient for local use.

Ideally, all hosts that are on the same physical network and are to exchange packets directly should have the same network number. However, hosts on different physical networks can also have the same network number. In this case, part of the local address can be used locally as a ***sub-network number*** and all host interfaces to the same physical network are given the same sub-network number.

A ***route*** defines a path for sending packets through the Internet network to an address on another network. A route does not define the complete path; it defines only the path segment from one host to a gateway host that can forward packets to a destination. A route can be defined as one of the following types:

host route Defines a gateway that can forward packets to a specific host on another network.

network route Defines a gateway that can forward packets to any of the hosts on a specific network.

default route Defines a gateway to use when a host or network route to a destination is not otherwise defined.

Routes are defined in an internal ***routing table***, which can hold up to 32 route definitions. These route definitions include information on networks reachable from the local host, gateways that can be used to reach remote networks, and the ***hop count*** (or distance metric) to those networks. The routing table is a dynamic structure maintained in the kernel by the Interface Program and can be updated manually.

Host processes originate and receive network packets. High-level processes generally do so by sending data to a protocol routine or process. The protocol blocks this data (and possibly ***fragments*** it into smaller blocks than it received) and adds a ***header***, which contains protocol control information for the data block. This protocol may then send the packet to a lower-level protocol, which adds its header (and may further fragment the data). This combination of a header and a data block is a packet. Note that a packet for a higher-level protocol can become the data for a lower-level protocol.

Eventually, the packets are sent to the IP process, which transmits the packets through the physical network to a connecting host. These packets may pass through several gateway hosts before they reach the IP process at the destination host. At the destination, the packet headers are stripped, and the packet data (possibly ***reassembled*** into larger higher-level packets) is sent to the appropriate protocol or process.

Protocols at different levels in the networks, gateways, and hosts support interprocess communication, providing one-way or two-way data flow on logical connections between processes on different hosts. A process in the Internet environment can have logical connections to several other processes and, if necessary, can treat each of the connected processes independently.

Each logical connection in the Internet environment is assigned to one port (or socket) at one end of the connection and one at the other end. A process may have one port used for all connections, or it may have a number of ports, with each port assigned to only one connection. To keep track of these ports, each Internet port on a host has an ***Internet port number*** that is unique for that host.

A particular port on a particular host is identified with an Internet ***socket address***, which is the combination of an Internet address and a port number. Since every Internet address is unique and every active port on a host has a unique port number, every socket address is also unique. This also means that every connection between Internet ports is uniquely identified since both ends of the logical connection are uniquely identified.

While IP handles only transporting individual packets, higher-level connection-level protocols create and maintain these connections between Internet ports (or sockets). The Interface Program supports two types of logical connections between hosts: datagram connections and stream connections.

The first of these, a ***datagram*** connection, is supported by the User Datagram Protocol (UDP) and is transaction-oriented (that is, each packet is sent as an individual transaction or message). Since UDP is designed to be a very simple and quick protocol, it does not provide for reliable delivery of packets. Any reliability must be provided by higher-level

protocols. For example, the Trivial File Transfer Protocol (TFTP) uses UDP for file transfers and provides a degree of reliability by requiring acknowledgement messages for each block of data sent to a destination.

The second type of logical connection, a ***stream*** connection, is supported by the Transmission Control Protocol (TCP) and allows a message to span more than one packet. This protocol provides for asynchronous data flow between hosts with full flow control by the destination while ensuring that data is delivered correctly, in sequence, and completely. However, this reliability is provided at the cost of more protocol overhead.

Application programs that use the Interface Program can send packets at the IP level and can create and manage connections at the UDP, TCP, or socket level. The following section describes how the Interface Program is implemented.

Structure of the Interface Program

Figure 1-1 on page 1-8 provides an overview of the relationship between the AIX TCP/IP Communications Subsystem, the kernel, and the VRM segments of the AIX Operating System. The figure portrays the principal Interface Program user and server commands, protocols, and Application Programming Interfaces (APIs).

The commands included with the Interface Program are shown at the top of Figure 1-1. These commands, as well as the TCP and UDP libraries, access the Internet protocols through routines available in the AIX sockets interface. Some of the Interface Program commands directly access the **pty** and **/dev/kmem** device drivers; however, these commands are exceptions. The commands that access the **/dev/kmem** device driver, for example, are used for configuration and management of the Interface Program, rather than for sending data across the network.

As Figure 1-1 shows, the Interface Program uses the AIX sockets interface, which contains the Internet Protocol (IP), Address Resolution Protocol (ARP), and Internet Control Message Protocol (ICMP). IP, which is the basic packet transport protocol for TCP and UDP, uses ARP for translating between Internet and hardware addresses. ICMP is a required protocol for any implementation of IP. ICMP handles status and error messages concerning IP packet transport. Note that AIX does not provide APIs to ARP and ICMP.

The sockets interface handles all IP and ARP packets sent to or received from the network. To do so, the interface uses the VRM Asynchronous Device Driver for serial, asynchronous communications through IBM Asynchronous Adapters, and uses the VRM Block I/O Manager for block I/O communications through IBM Baseband Adapters, IBM Token-Ring Adapters, and customer-provided X.25 Adapters and 802.3 Adapters.

Note: The API for the sockets interface, which provides the Interface Program with access to the protocols, is described in *AIX Operating System Technical Reference*. This publication documents only the APIs for the TCP and UDP libraries, which are available for compatibility with existing programs. The APIs for the Block I/O Device Manager and for the VRM device handlers are described in *VRM Device Support*.

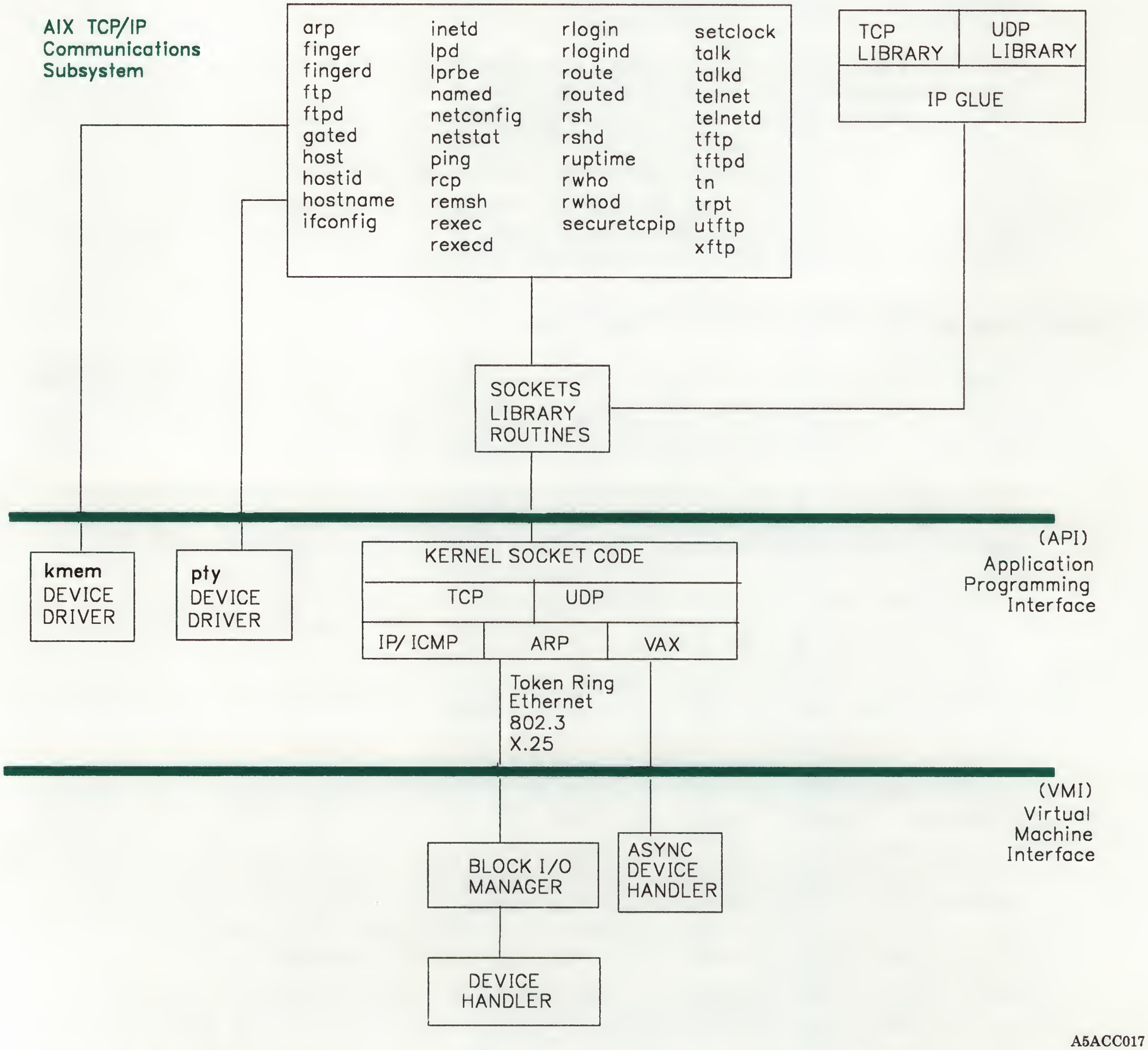


Figure 1-1. Interface Program for use with TCP/IP Commands, Protocols, and APIs

Commands

The Interface Program provides the following general types of commands:

- File transfer
- Remote mail and conversation
- Remote login, command execution, and printing
- Network management.

Following are brief explanations of the individual commands in each of these groups.

File Transfer

The Interface Program contains three file transfer commands: **ftp**, **rcp**, and **tftp** (**utftp**).

ftp or **xftp** The **ftp** command implements the File Transfer Protocol (FTP), which makes it possible to transfer data among similar or dissimilar hosts and to transfer data between foreign hosts indirectly. **ftp** can transfer files in either ***ascii*** format (that is, as 7-bit ASCII characters) or ***binary*** format (that is, as 8-bit binary data).

The **ftp** command provides subcommands for such tasks as listing foreign directories, changing the current local and foreign directory, transferring multiple files in a single request, creating and removing directories, and escaping to the local shell to perform shell commands. **ftp** also provides for security by sending passwords to the foreign host and permits automatic login, file transfers, and logout. See "File Transfer Protocol (FTP)" on page 1-19, "**ftp**, **xftp**" on page 2-9, and "**ftpd**" on page 3-5 for related information.

rcp The **rcp** command makes it possible to easily transfer data between similar hosts. The operation and syntax of the **rcp** command is similar to that of the **cp** command. See "**rcp**" on page 2-47 and "**rshd**" on page 3-33 for additional information.

tftp (utftp) The **tftp** command implements the Trivial File Transfer Protocol (TFTP). Its only function is to read and write files to and from a foreign host. **tftp** cannot list directories and it has no provisions for user authentication. **tftp** passes 8-bit bytes of data. **utftp** is a form of **tftp** for use in a pipe.

For related information, see "**tftp**, **utftp**" on page 2-82 and "**tftpd**" on page 3-43.

The Interface Program also provides services for the Basic Network Utilities file transfer commands and the **uucpd** server command, which are included in the Basic Network

Utilities and are not part of the Interface Program. Refer to *Using the AIX Operating System* and *Managing the AIX Operating System* for more information on these commands.

Remote Mail and Conversations

The Interface Program provides the **talk** command, which allows two users on the same host or different hosts to have an interactive conversation. **talk** opens both a send window and a receive window on each user's display. Each user is then able to type into the send window while **talk** displays what the other user is typing. See "**talk**" on page 2-70 and "**talkd**" on page 3-39 for more information.

The Interface Program also provides services for commands that are specifically designed for sending mail (messages) between users on different hosts. These mail-oriented commands include the **mail** and **sendmail** commands and the commands included with the Message Handling (MH) package.

The **mail** command supports sending and receiving mail (messages) between users on different hosts using host names. Normally, the **sendmail** server command is used for mail transfers between hosts using the Simple Mail Transfer Protocol (SMTP). Since the **mail** and **sendmail** commands are not a part of Interface Program, refer to *Using the AIX Operating System* for information on using the mail facilities and to *Managing the AIX Operating System* for information on managing mail. Also, see *AIX Operating System Commands Reference* for information on the **mail** and **sendmail** commands.

The Message Handling (MH) commands also support sending mail between hosts. These commands, which provide extensive services for processing mail, are documented in *Managing the AIX Operating System* and *AIX Operating System Commands Reference*.

Remote Login, Command Execution, and Printing

The Interface Program contains the **lprbe** command for printing at a foreign host, the **rsh** and **rexec** commands for executing commands at a foreign host, and the **rlogin** and **telnet** commands for logging on to a foreign host.

lprbe The **lprbe** command (a backend program for the print spooler) allows the local host to send a print job to the printer on the foreign host. See "**lprbe**" on page 2-31 and "**lpd**" on page 3-16 in this book, and the **print** command in *AIX Operating System Commands Reference* for additional information.

rexec The **rexec** command makes it possible to execute commands on similar foreign hosts without maintaining a TELNET session. For additional information, see "**rexec**" on page 2-51 and "**rexecd**" on page 3-24.

rlogin The **rlogin** command makes it possible to log in on similar foreign hosts. For additional information on using the **rlogin** command, see "**rlogin**" on page 2-54 and "**rlogind**" on page 3-26.

rsh or **remsh** The **rsh** command makes it possible to execute commands on similar foreign hosts. For additional information, see "**rsh**, **remsh**" on page 2-60 and "**rshd**" on page 3-33.

telnet The **telnet** command is a terminal emulation program that allows you to log in on a similar or dissimilar foreign host. It implements the Telnet Protocol (TELNET), which is a bi-directional, 8 bit-per-byte communications facility and provides an interface between terminal devices and terminal-oriented processes. See "**telnet**, **tn**" on page 2-72 and "**telnetd**" on page 3-41 for related information.

Network Management

The Interface Program contains commands for determining the status of hosts and users on the network and for configuring and managing the Interface Program.

arp The **arp** command displays or changes the Internet address to hardware address translation tables used by the Address Resolution Protocol. See "**arp**" on page 2-3 for additional information.

finger The **finger** command returns information about users on a specified host. See "**finger**" on page 2-6 and "**fingerd**" on page 3-3 for additional information.

host The **host** command shows the Internet address of a specified host. It first sends requests to name servers in the network, then searches for the address in a small, local table. See "**host**" on page 2-22 for additional information.

hostid The **hostid** command sets or displays the identifier of the local host. See "**hostid**" on page 2-24 for additional information.

hostname The **hostname** command shows or sets the Internet name and address of the local host. See "**hostname**" on page 2-26 for additional information.

ifconfig The **ifconfig** command configures network interfaces and their characteristics. See "**ifconfig**" on page 2-28 for additional information.

netconfig The **netconfig** command configures the adapter or adapters TCP is to run across according to the information in the **/etc/net** file. See "**netconfig**" on page 2-35 and "**net**" on page 4-52 for additional information.

netstat The **netstat** command shows local and foreign addresses, routing tables, hardware statistics, and summary of packets transferred. See "**netstat**" on page 2-39 for additional information.

ping The **ping** command sends an echo request to a network host to determine whether that host is operational and on the network. See "**ping**" on page 2-44 for additional information.

route — The **route** command permits you to manually manipulate the routing tables. See "**route**" on page 2-57, "**gated**" on page 3-9, and "**routed**" on page 3-28 for additional information.

ruptime — The **ruptime** command shows status information on hosts that are connected to local physical networks and are running the **rwhod** server. See "**ruptime**" on page 2-63 and "**rwhod**" on page 3-35 for more information.

rwho — The **rwho** command shows status information for users on hosts that are connected to local physical networks and are running the **rwhod** server. See "**rwho**" on page 2-65 and "**rwhod**" on page 3-35 for more information.

setclock — The **setclock** command reads the network time service and sets the time and date of the local host accordingly. See "**setclock**" on page 2-68 for additional information.

trpt — The **trpt** command performs protocol tracing on sockets. See "**trpt**" on page 2-89 for additional information.

Additionally, the Interface Program contains server commands (or ***daemons***), which provide additional facilities for configuring and managing the network. Refer to Chapter 3, "Server Commands" for details on these commands.

Protocols

AIX provides the protocols that are required to comply with RFC 1011, *Official Internet Protocols*, along with other protocols that are commonly used by hosts in the Internet community.

Note: The usage of Internet network, version, socket, service, and protocol numbers in AIX and the Interface Program also complies with RFC 1010, *Assigned Numbers*.

The AIX sockets interface provides the following Internet network-level protocols:

- Internet Protocol (IP)
- Address Resolution Protocol (ARP)
- Internet Control Message Protocol (ICMP)
- VAX[1] Trailer Encapsulation Protocol (VAX).

Note: ICMP uses the basic support of IP as if it were a higher-level protocol. However, ICMP is actually an integral part of IP and must be implemented by every IP module.

The sockets interface also provides the following Internet transport-level protocols:

- User Datagram Protocol (UDP)
- Transmission Control Protocol (TCP).

Applications within the Interface Program provide various other higher-level Internet protocols, including:

- Domain Name Protocol (DOMAIN)
- Exterior Gateway Protocol (EGP)
- File Transfer Protocol (FTP)
- Telnet Protocol (TELNET)
- Trivial File Transfer Protocol (TFTP).

Other applications within the Interface Program provide other higher-level protocols, which are not official Internet protocols but are commonly used in Internet networks. These protocols include:

- DCN Local Network Protocol (HELLO)
- Remote Login Protocol (LOGIN)
- Remote Printing Protocol (PRINTER)

---

1 VAX is a trademark of Digital Equipment Corporation.

- Remote Shell Protocol (SHELL)
- Remote Command Execution Protocol (EXEC)
- Routing Information Protocol (RIP).

The following are brief explanations of most of these protocols. Additional information on these and other protocols is contained in Chapter 2, "User Commands" and Chapter 3, "Server Commands."

Internet Network-Level Protocols

The IP protocol is the basic network-level protocol for sending packets between Internet hosts. IP is used by all higher-level protocols. AIX also provides the following protocols to support other network-level tasks:

ARP Used for translating between IP addresses and physical network addresses.

ICMP Used for handling status and control messages concerning IP operations. There is no API to ICMP; it must be accessed through IP.

VAX Used for handling VAX packets that have postfixed control information.

Since these protocols are generally not used directly by higher-level protocols and applications, no APIs are provided for them.

Internet Protocol (IP)

The Internet Protocol (IP) provides the interface from the transport-level (host-to-host) protocols to the physical-level protocols. IP is used by transport-level protocols in an Interface Program environment as the basic transport mechanism for sending IP packets from one host to another. IP uses local area network, wide area network, and serial line protocols to carry packets (or IP datagrams) to the next gateway or destination host. Addressing at this level is usually from host to host.

The IP is designed for use in interconnected systems of packet-switched networks. IP provides the means to transmit blocks of data (or packets of bits) from sources to destinations. Sources and destinations are hosts identified by fixed-length addresses. Outgoing packets automatically have an IP header prefixed to them and incoming packets have their IP header removed before being sent to the higher-level protocols. This protocol provides for the universal addressing of hosts in the Internet network.

IP, however, does not provide a reliable communication facility because it does not require acknowledgements from the sending host, the receiving host, or intermediate hosts. IP also does not provide error control for data; it provides only a header checksum. IP treats each datagram as an independent entity unrelated to any other datagram. It does not perform retransmissions or flow control. A higher-level protocol that uses IP must implement its own reliability procedures if it requires reliable communications.

The AIX implementation of IP complies with the requirements of the IP specification, RFC 791, *Internet Protocol.* The type of service defaults to *routine precedence*, *normal delay*, *normal throughput*, and *normal reliability*. The total length of IP packets can be configured independently for each interface, and packet fragmentation at gateways and reassembly at destinations is provided. The IP packets are given a default time to live of 255 seconds. AIX implements the following IP architecture options:

- End of Option List
- No Operation
- Security (user-definable with the **ip_security** keyword in **/etc/net**)
- Loose Source and Record Route
- Strict Source and Record Route
- Record Route
- Stream Identifier
- Internet Timestamp.

The AIX implementation of IP complies with the IP address specification in RFC 1010, *Internet Assigned Numbers*, including the special addresses for addressing and broadcasting to directly connected networks and hosts and for local loopback. Refer to "IP Addresses" on page 1-29 for more information on IP addressing.

The following IP parameters can be set on an interface-by-interface basis using entries in the **/etc/net** file. The Internet address and subnet address mark can also be set by issuing the **ifconfig** command with the appropriate parameters.

- Internet address
- Subnet address mask
- Maximum local packet length
- Maximum remote packet length.

The programming interface to IP is through calls to the sockets library routines, which are described in *AIX Operating System Technical Reference.*

Address Resolution Protocol (ARP)

The Address Resolution Protocol (ARP) dynamically maps Internet addresses and hardware addresses on local area networks. The Interface Program uses ARP to collect and distribute the information for ***mapping tables*** that map Internet addresses into addresses for the following adapters: IBM Baseband Adapter, IBM Token-Ring Adapter, or customer-provided 802.3 Adapter. ARP mapping tables are not used for X.25 Adapters.

The kernel maintains the mapping tables. ARP is not directly available to users or applications. When an application sends an Internet packet to one of the interface drivers,

the driver requests the appropriate address mapping. If the mapping is not in the mapping table, an ARP broadcast packet is sent through the requesting interface driver to the hosts on the local area network.

When any host that supports ARP receives an ARP request packet, the host notes the IP and hardware addresses of the requesting system, and updates its mapping table if necessary. If the receiving host IP address does not match the requested address, the host discards the request packet. If the IP address does match, the receiving host sends a response packet to the requesting system. The requesting system caches the new mapping and transmits any pending Internet packets.

Internet Control Message Protocol (ICMP)

When a gateway or destination host occasionally needs to communicate directly with a source host, the Internet Control Message Protocol (ICMP) is used. For example, ICMP messages may be sent in any of the following situations:

- When a packet cannot reach its destination
- When a gateway does not have the buffering capacity to forward a packet
- When a gateway can direct a host to send traffic on a shorter route.

The Interface Program sends and receives the ICMP message types listed below:

echo request	**echo reply**
information request	**information reply**
timestamp request	**timestamp reply**
address mask request	**address mask reply**
destination unreachable	**parameter problem**
source quench	**time exceeded**
redirect message	

Note: The purpose of ICMP is to provide feedback about problems in the communication environment, not to make IP reliable. The use of ICMP does not guarantee that an IP packet will be delivered reliably or that an ICMP message will be returned to the source host when an IP packet is not delivered or incorrectly delivered.

ICMP is embedded in the **ping** user command and the kernel, and no API is provided to this protocol. Refer to "**ping**" on page 2-44 for additional information.

VAX Trailer Encapsulation Protocol

VAX Trailer Encapsulation Protocol (the protocol that supports VAX trailers) moves all variable length header information in an IP packet to a position following the data segment. Trailer encapsulation allows the receiving host to receive data on a page-aligned boundary, a requirement for exploiting a page-mapped virtual memory environment. The RT system receives and processes VAX trailer protocol data and can be configured to send it, using the **ifconfig** command.

Internet Transport-Level Protocols

UDP and TCP are the basic transport-level protocols for making host-to-host connections between Internet hosts. Higher-level protocols and applications use UDP to make datagram connections and TCP to make stream connections. The AIX sockets interface implements these protocols.

User Datagram Protocol (UDP)

The User Datagram Protocol (UDP) provides a datagram mode of communication between Internet hosts. This protocol assumes that IP is the underlying protocol. UDP provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. Like IP, UDP is transaction-oriented and offers no assurance of datagram delivery or duplication protection. However, UDP adds the ability to specify source and destination port numbers and provides checksums for both header and data portions of the datagram.

AIX implements UDP as described in RFC 768, *User Datagram Protocol.* The API to UDP consists of a set of library calls provided by the sockets interface. An API for existing programs is also provided by the UDP library. Refer to the description of the sockets routines in *AIX Operating System Technical Reference* and to "**udp**" on page C-18 in this book for more information on these APIs to UDP.

Note: Applications that require reliable delivery of datagrams must implement their own reliability checks when using UDP. Applications that require reliable delivery of streams of data should generally use TCP.

Transmission Control Protocol (TCP)

The Transmission Control Protocol (TCP) provides a reliable, stream mode of communication between Internet hosts. TCP assumes that IP is the underlying protocol and supports the block transmission of a continuous stream of 8-bit bytes between process ports. TCP is designed to ensure that data is not damaged, lost, duplicated, or delivered out of order to a receiving process. The receiving process controls the rate at which it receives the data.

AIX implements TCP in accordance with Department of Defense Standard Transmission Control Protocol as described in RFC 793, *Transmission Control Protocol.* The API to TCP consists of a set of library calls provided by the sockets interface. An API for existing

programs is also provided by the TCP library. Refer to the description of the sockets routines in *AIX Operating System Technical Reference* and to "**tcp**" on page C-13 in this book for more information on these APIs to TCP. Refer to RFC 793, RFC 813, RFC 814, RFC 816, RFC 817, RFC 879, RFC 889, RFC 896, and MIL-STD-1778 for details on TCP.

All three of the TCP architecture options are implemented:

- END OF OPTION LIST
- NO OPERATION
- MAXIMUM SEGMENT SIZE.

The TCP retransmission timeout value is dynamically determined for each connection, based on round-trip time.

TCP implementation allows a segment size of at least 1024 bytes.

Specialized Internet Protocols

The Interface Program implements higher-level Internet protocols at the application program level, but does not provide APIs to these protocols. Some of these protocols are described in the next pages.

Domain Name Protocol (DOMAIN)

The Domain Name Protocol (DOMAIN) allows one or more hosts to act as a name server for other hosts within a domain, assuming that UDP is the underlying protocol. DOMAIN allows the local network to assign host names in their domain independently from names in other domains, making DOMAIN the preferred Internet name resolution protocol.

In a DOMAIN environment, local resolver routines either resolve Internet names and addresses, using a local name resolution database maintained by a **named** process, or they request name resolution services from a remote DOMAIN name server host. In either case, if the name resolution information is unavailable, the routines attempt to use the **/etc/hosts** file for name resolution.

Note: The Interface Program configures local resolver routines for the DOMAIN protocol if the local file **/etc/resolv.conf** exists. If this file does not exist, the Interface Program configures the local resolver routines to use the **/etc/hosts** database.

The Interface Program implements the DOMAIN protocol in the **named** server command and in the kernel resolver routines and does not provide an API to this protocol. For more information on the **named** server, refer to "**named**" on page 3-20. For more information on the **/etc/resolv.conf** and **/etc/hosts** files, refer to "**resolv.conf**" on page 4-64 and "**hosts**" on page 4-24. For more information on the kernel resolver routines, refer to the description of the **gethostbyaddr** subroutines in *AIX Operating System Technical Reference*. For more information on the DOMAIN protocol, refer to RFC 1032, RFC 1033, RFC 1034, and RFC 1035.

Exterior Gateway Protocol (EGP)

The Exterior Gateway Protocol (EGP) allows a host to act as a gateway between networks that do not need to share detailed information about routing within the respective networks. In contrast to the Routing Information Protocol (RIP), which can be used within an ***autonomous system*** of Internet networks that dynamically reconfigure routes, EGP routes are predetermined in the **/etc/gated.conf** file. EGP assumes that IP is the underlying protocol.

The Interface Program implements the EGP protocol in the **gated** server command and does not provide an API to this protocol. For more information on **gated** and the **/etc/gated.conf** file, refer to "**gated**" on page 3-9 and "**gated.conf**" on page 4-7. For more information on EGP, refer to RFC 823, RFC 888, RFC 890, and RFC 904.

File Transfer Protocol (FTP)

The File Transfer Protocol (FTP) is designed to make it possible to transfer data among dissimilar hosts and to transfer files between two foreign hosts indirectly. FTP assumes that TCP is the underlying protocol, uses a TELNET connection to transfer commands and replies, and uses an FTP data connection to transfer files as either 7-bit netascii characters or as 8-bit binary data.

FTP provides for such tasks as listing remote directories, changing the current remote directory, creating and removing remote directories, and transferring multiple files in a single request. FTP provides for security by passing user and account passwords to the foreign host.

The Interface Program implements FTP in the **ftp** user command and the **ftpd** server command and does not provide an API to this protocol. Refer to "**ftp**, **xftp**" on page 2-9 and "**ftpd**" on page 3-5 for more information and to RFC 765 and RFC 959 for more information on FTP.

AIX implements a user-server and a server-server relationship through the **ftp** command. **ftp** provides the server with the following configuration:

TYPE	ASCII Binary Form non_print
MODE	Stream
STRU	File
COMM	MODE, NOOP, PORT, QUIT, RETR, STOR, STRU, TYPE, USER

Default values for transfer parameters are:

TYPE	ASCII Form non_print
MODE	Stream
STRU	File

Telnet Protocol (TELNET)

The Telnet Protocol (TELNET) provides a standard method to interface terminal devices and terminal-oriented processes to each other. TELNET assumes that TCP is the underlying protocol, provides for full-duplex (concurrently bi-directional) communication and sends data as either 7-bit netascii characters or as 8-bit binary data.

TELNET is commonly used by terminal emulation programs that allow you to log in to a foreign host. However, TELNET can also be used for terminal-to-terminal communication (called ***linking***) and inter-process communication (called ***distributed computation***). TELNET is also used by other protocols (for example, FTP) for establishing a protocol control channel.

The Interface Program implements TELNET in the **telnet** and **tn** user commands and the **telnetd** server command and does not provide an API to TELNET. Refer to "**telnet, tn**" on page 2-72 and "**telnetd**" on page 3-41 for more information on the **telnet**, **tn**, and **telnetd** commands. Refer to RFC 854 for more information on TELNET and to RFC 855 for more information on TELNET options.

The following TELNET options are supported:

- BINARY TRANSMISSION (used in telnet3270 sessions)
- ECHO (a user-changeable command)
- SUPPRESS GO_AHEAD (The RT system suppresses go_aheads.)
- SAK (Secure Attention Key)
- TIMING MARK (recognized, but has a negative response)
- EXTENDED OPTIONS LIST (recognized, but has a negative response).

Trivial File Transfer Protocol (TFTP)

The Trivial File Transfer Protocol (TFTP) is designed only to read and write files to and from a foreign host. Since TFTP assumes that UDP is the underlying protocol, it is generally quicker than FTP. Like FTP, TFTP can transfer files as either 7-bit netascii characters or as 8-bit binary data. However, unlike FTP, TFTP cannot be used to list or change directories at a foreign host, and it has no provisions for user authentication.

The Interface Program implements TFTP in the **tftp** and **utftp** user commands and in the **tftpd** server command. **utftp** is a form of **tftp** for use in a pipe. The Interface Program does not provide an API to this protocol. Refer to "**tftp, utftp**" on page 2-82, and "**tftpd**"

on page 3-43 for more information on these commands. For more information on TFTP, refer to RFC 783.

Other Network Protocols

The Interface Program implements other higher-level protocols that are not official Internet protocols but are commonly used in the Internet community at the application program level. The Interface Program does not provide APIs to these protocols. Some of these protocols are described in the following paragraphs.

DCN Local-Network Protocol (HELLO)

The **gated** server command provides the DCN local-network protocol, which maintains connectivity, routing, and timekeeping information. For information about **gated** and the HELLO protocol, refer to "**gated**" on page 3-9.

Remote Command Execution Protocol (EXEC)

The **rexec** user command and the **rexecd** server command provide the remote command execution protocol, which allows users to execute commands on a compatible foreign host. For information about **rexec**, **rexecd**, and the remote command execution protocol, refer to "**rexec**" on page 2-51 and "**rexecd**" on page 3-24.

Remote Login Protocol (LOGIN)

The **rlogin** user command and the **rlogind** server command provide the remote login protocol, which allows users to log in to a foreign host and use their terminals as if they were directly connected to the foreign host. For information about **rlogin**, **rlogind**, and the remote login protocol, see "**rlogin**" on page 2-54 and "**rlogind**" on page 3-26.

Remote Printing Protocol (PRINTER)

The **lprbe** backend print spooler command and the **lpd** server command provide the remote printing protocol, which allows a host to act as a remote print server for other hosts. For information about **lpd**, **lprbe**, and the remote printing protocol, see "**lprbe**" on page 2-31 and "**lpd**" on page 3-16.

Remote Shell Protocol (SHELL)

The **rsh** user command and the **rshd** server command provide the remote command shell protocol, which allows users to open a shell on a compatible foreign host for executing commands. For information about **rsh**, **rshd**, and the remote command shell protocol, refer to "**rsh, remsh**" on page 2-60 and "**rshd**" on page 3-33.

Routing Information Protocol (RIP)

The routing daemons (**routed** and **gated**) use a variant of the Xerox[2] NS Routing Information Protocol to dynamically keep track of routes and maintain kernel routing table entries. For more information on **routed** and RIP, see "**routed**" on page 3-28. For more information on **gated** and RIP, see "**gated**" on page 3-9.

---

2 Xerox is a trademark of Xerox, Inc.

Internet Router

The Internet Router enables an Interface Program host to act as a gateway for routing data between separate networks that use any of the following adapters:

- IBM RT Baseband Adapter for use with Ethernet or 802.3
- IBM Token-Ring Network RT Adapter
- 802.3 Adapter (customer-supplied)
- X.25 Adapter (customer-supplied).

Modems, which provide a serial line connection between hosts, can also be used.

Depending on slot availability, you can install one or two Baseband Adapters, one to four IBM Token-Ring Adapters, one to four 802.3 Adapters, and one X.25 Adapter in the RT system. The adapters (or interfaces) are defined by adding one entry for each adapter to each of the following files:

/etc/net
/etc/hosts.

Once an interface is configured, some options can be redefined with the **ifconfig** command.

Each host has one primary name but, because of the entries in **/etc/hosts**, can have multiple secondary names, which are used in routing. References to gateway hosts can be by primary or secondary name.

Figure 1-2 represents five networks and four gateway hosts, and it demonstrates how routing can be designed. The networks are:

200	(`host1`, `host2`, `host3`, `host4` `host5`, `host6`, `host7`, `host8`, `host9`, `host10`, `host11`, and `host12`)
201	(`host5`, `host13`, `host14`, and `host15`)
202	(`host15`, `host16`, and `host17`)
203	(`host9`, `host18`, and `host19`)
204	(`host16` and `host20`)

Hosts `host5` and `host15` are the gateway hosts.

Routes can be established in either of two ways:

- Explicitly, by setting up specific hosts in routing tables
- Dynamically, by running either **routed** or **gated** (the routing daemons) on the gateway hosts.

The routing daemons (**routed** and **gated**), which can be run on a gateway host, query other defined gateway hosts periodically for the information necessary to generate, update, and maintain routing tables. **routed** uses two files, **/etc/gateways** and **/etc/networks**, to

determine with which hosts routing information should be exchanged. **gated** uses the **/etc/gated.conf** file to determine which types of routing to perform, as well as which options to use. For more information on **routed**, see "**routed**" on page 3-28, "**gateways**" on page 4-21, and "**networks**" on page 4-58. For more information on **gated**, see "**gated**" on page 3-9 and "**gated.conf**" on page 4-7.

In Figure 1-2, gateway `host5` contains two interfaces: one IBM Token-Ring Adapter and one IBM Baseband Adapter. Gateway `host15` contains four interfaces: one IBM Baseband Adapter and three direct serial connections. Gateway `host9` contains two interfaces: one IBM Token-Ring Adapter and one 802.3 Adapter, while gateway `host16` contains a direct serial connection and an X.25 Adapter connection.

Because `host5` in this example is a gateway host, it is an appropriate host to designate as both the network name server and time server. Such a host can be reached by all other hosts on the network. In addition, a gateway host is frequently available 24 hours per day. All other hosts on the network should identify `host5` with the **nameserver** entry in their **/etc/resolv.conf** files.

In the Figure 1-2 configuration, routing could be established for `host1` as follows: Decide on a default gateway. Any packets sent to a host that is not on a network go to the default gateway for that network. To establish `host5` as the default gateway on the network that includes `host5` and `host1`, enter the following **route** command on `host1`:

```
route add 0 host5
```

This route command establishes `host5` as the exit point, or default gateway, from that network. (For more detail on using the **route** command, see "**route**" on page 2-57.)

On the `201` network, there are two gateways (`host5`, the default gateway, and `host15`). The preferred routing is to the default gateway.

Note: This approach to routing works only if the gateway host can send ICMP redirect messages, which the RT system can do.

An alternate way to establish routing to the `200` network from the `202` network is to set up routes explicitly with **route** commands like the following ones:

```
route add 192.9.200.0 host5
route add 192.9.202.0 host15
```

If the route command is used for routing, the routing table must be configured on both networks. For example, for `host1` on network `192.9.200.0` to have access to all hosts on network `192.9.201.0`, `host1` must issue the **route** command and all hosts on the `192.9.201.0` network must issue the **route** command to access the `192.9.200.0` network.

The size of data packets transmitted through the gateway host may vary, which can affect performance. TCP data transmitted to a gateway host that is not on the same network can

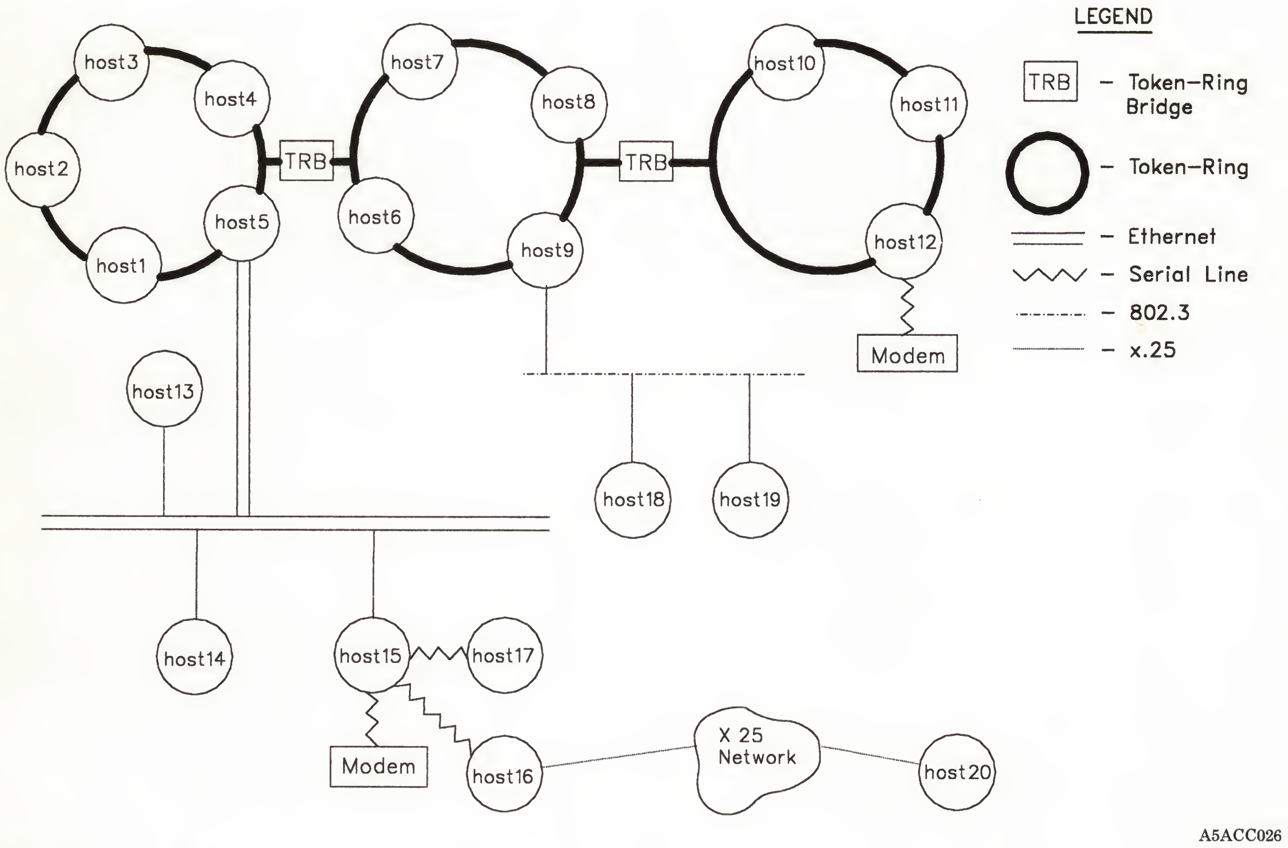


Figure 1-2. Network and Gateway Routing

be sent in 576-byte packets to avoid fragmentation and reassembly of the packets or, if the gateway can handle packets larger than 576-bytes, **r_inetlen** (remote **inetlen**) can be set to the larger packet size. **r_inetlen** is set on an individual interface basis and not per host or network. If **r_inetlen** is not specified in the **/etc/net** file, it is set to 576.

The IBM Token-Ring Adapter supports both gateways and bridges. With bridging, the routing information field can contain addresses that are up to eight hops away from the sending host. The minimum frame size that can be passed through the bridge is 1K bytes. If the frame size is 1K bytes, **inetlen** should be set to 576. Otherwise, **inetlen** can be set to 1064 or 1568 bytes.

The maximum packet size for an X.25 Adapter is 1007 bytes. For an 802.3 Adapter, the maximum size is 1500 bytes.

The user can specify whether all rings in a Token-Ring Network can communicate or just those within the local ring using **localbroadcast**. To allow machines on different rings on the same Token-Ring Network to communicate, **localbroadcast** is set to false. Setting **localbroadcast** to **true** allows only those machines on the same ring to communicate. **localbroadcast** is set on an individual interface basis and not per host or network. If **localbroadcast** is not specified in the **/etc/net** file, it is set to **true**.

Both **r_inetlen** and **localbroadcast** are used to help improve performance by allowing packet transmission without fragmentation and only the necessary requests and broadcasts to occur.

Names and Addresses

As a convenience to users, the Internet protocols provide for the use of symbolic names. However, at the protocol level, these names must be resolved (translated) into Internet addresses. This section explains how the Interface Program resolves Internet names into Internet addresses and the Internet conventions for assigning names and addresses to hosts on the network.

Name Resolution

When a process receives a symbolic name and needs to resolve it into an address, it calls a kernel ***resolver routine***. The method used by the routine to resolve names depends on how the the local host is configured. This, in turn, depends on whether the local network is organized either as a ***flat network*** or as a ***domain network***.

In a flat network, all hosts in the network are administered by one central authority. This form of network requires that all hosts in the network must have unique host names, and in a large network, creates a large administrative burden on the central authority. This is how the ARPA Network (ARPANET) and the Internet community in general used to be organized. However, the administrative burden became so great that the Internet community reorganized into domain networks.

In a domain network, groups of hosts are administered separately within a tree-structured hierarchy of domains and subdomains. In this case, host names need to be unique only within the local domain, and only the ***root domain***, which is made up of subdomains (and possibly a few hosts), is administered by a central authority. This structure allows subdomains to be administered locally and reduces the burden on the central authority.

Unfortunately, the local administrative burden can be much higher in a domain network than in a flat network. Additionally, this burden may be considered excessive in a small network that will never connect to the Internet community. Therefore, the AIX resolver routines and the Interface Program are designed to operate in either type of network.

As stated previously, the way the network is organized determines how resolver routines handle name resolution. Additionally, this determines the method used by resolver routines when communicating with remote ***nameserver*** hosts (that is, hosts that resolve names for other hosts).

The resolver routines and the Interface Program determine which type of network they are dealing with by whether the **/etc/resolv.conf** file exists. If the file exists, they assume that the local network is a domain network. Otherwise, they assume that the local network is a flat network.

To resolve a name in a flat network, the resolver routine checks the **/etc/hosts** file for an entry that maps the name to an address.

To resolve a name in a domain network, the resolver routine first queries the domain nameserver database, which may be local if the host is a domain name server or may be on

a foreign host. If the routine is using a remote name server, the routine uses the Domain Name Protocol (DOMAIN) to query for the mapping. If the local or remote query fails, the routine then checks for an entry in the local **/etc/hosts** file.

Internet Names

The Internet Protocol provides standards for using symbolic names for hosts and domains. These standards and the methods used by the Interface Program for establishing these names are described in the following.

Host Names

As a convenience to users, each host on an Internet network is assigned one or more host names. In the Interface Program implementation, these names are a maximum of 32 case-insensitive characters in length and cannot contain embedded blanks. Depending on how the local network resolves host names into Internet addresses, these host names are associated with one or more Internet addresses.

The **hostname** command must be run to identify the official local host name and the primary IP address to the local Interface Program. **hostname** takes the name of the host to be identified as a parameter and sets the standard host name. Since the **hostname** command makes a temporary change to the local host name that is identified to IP, this command is typically run at system start by an entry in the **rc.tcpip** command file. A host name to be set by **hostname** must appear in the **/etc/hosts** file, which associates local host names and addresses.

Warning: The name for a particular host must be specified the same way in both of the following places: following the **hostname** command in the **/etc/rc.tcpip** file, and on the line in the **/etc/hosts** file that contains the Internet address of the local host.

If these names do not match, unpredictable results may occur.

In a DOMAIN name server environment, the host name set in the **/etc/hosts** file must be the official name of the host, as returned by the name server. Generally, this name is the full domain name of the host in the form: *host.subdomain.subdomain.rootdomain*.

When using a DOMAIN name server in conjunction with the **sendmail** program, the **sendmail** configuration file, **/usr/adm/sendmail/sendmail.cf**, must be edited to reflect this official host name. In addition, the domain name macros in this configuration file must be set for **sendmail** to operate correctly. For more information on using **sendmail** with a DOMAIN name server, see the section on managing the mail systems in *Managing the AIX Operating System*.

For more information on the **hostname** command and the **/etc/hosts** file, refer to "**hostname**" on page 2-26 and "**hosts**" on page 4-24. For more information about **rc.tcpip**, refer to "**rc.tcpip**" on page 4-62.

Domain Names

In a domain network, each domain is given a local domain name. The DOMAIN protocol specifies that a local domain name must be less than 64 characters in length and that a fully specified domain name for a host is specified with the host name given first, followed by a period (.), followed by a series of local domain names separated by periods, and ending with the root domain, which is specified by a trailing period. Additionally, a fully specified domain name for a host (including periods) must be less than 255 characters in length. However, if a foreign host is in the local domain, the host can be specified with just the host name.

For a host that is in a domain network but is not a name server, the local domain name and the domain name server are specified in the **/etc/resolv.conf** file. In a DOMAIN name server host, the local domain and other name servers are defined in files read by the **named** daemon when it starts. For additional information, refer to "**resolv.conf**" on page 4-64 and "**named**" on page 3-20.

Internet Addresses

The Internet Protocol provides standards for assigning addresses to networks, sub-networks, hosts, and sockets and for using special addresses for broadcasts and local loopback. These standards and the methods used by the Interface Program for establishing these addresses are described in the following sections. For details on specific address assignments, refer to RFC 1010.

IP Addresses

The Internet Protocol (IP) uses a two-part, 32-bit ***address field***. The first part of the address field contains the network address; the second part contains the local address. There are four different types of address fields (class A, B, C, D), depending upon how the bits are allocated.

Figure 1-3 represents a class A address. It has a 7-bit network number and a 24-bit local address. The highest-order bit is set to 0. Since a network number of all 0's is reserved to broadcasts to the local network, and network number 127 is reserved for local loopback, there are 126 possible class A networks.

0	1 2 3 4 5 6 7	1 2 3 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
0	Network	Local Address

Figure 1-3. Class A Address

Figure 1-4 represents a class B address. It has a 14-bit network number and a 16-bit local address. The highest-order bits are set to 1 and 0. There are 16,384 possible class B networks.

0 1	1 2 3 4 5 6 7 8 9 0 1 2 3 4 5	2 3 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
1 0	Network	Local Address

Figure 1-4. Class B Address

Figure 1-5 represents a class C address. It has a 21-bit network number and an 8-bit local address. The three highest-order bits are set to 1, 1, and 0. There are 2,097,152 possible class C networks.

0 1 2	1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3	3 4 5 6 7 8 9 0 1
1 1 0	Network	Local Address

Figure 1-5. Class C Address

Notes:

1. Class C addresses starting at 192 are recommended for use with the Interface Program.
2. There is a set of reserved network addresses (for example, for accessing ARPA).
3. The Interface Program does not implement multicast addresses with the highest-order bits set to 1-1-1. These addresses (sometimes called class D) are reserved.
4. No addresses are allowed with the highest-order bits set to 1-1-1-1. These addresses (sometimes called class E) are reserved.

A commonly used notation for Internet host addresses is the ***dotted decimal***, which divides the 32-bit address into four 8-bit fields. The value of each field is specified as a decimal number and the fields are separated by periods (for example, 010.002.000.052, or 10.2.0.52).

Examples in this publication use the dotted decimal notation in the following forms:

Class A *nnn.lll.lll.lll*

Class B *nnn.nnn.lll.lll*

Class C *nnn.nnn.nnn.lll*

where *nnn* represents part or all of a network number and *lll* represents part or all of a local address.

The Interface Program determines local Internet addresses on an interface-by-interface basis. These addresses are determined by entries in the **/etc/net** file, which must agree

with entries in the **/etc/hosts** file. Refer to "**net**" on page 4-52 and "**hosts**" on page 4-24 for more information on these files.

Broadcast Addresses

The Interface Program can send data to all hosts on a local network, or to all hosts on all directly connected networks. Such transmissions are called ***broadcast messages***. For example, the routing daemons (**routed** and **gated**) use broadcast messages to maintain routing tables.

For data to be broadcast to all hosts on all directly connected networks, UDP and IP are used to send the data, and the destination address in the IP header is set to 0xFFFFFFFF. For data to be broadcast to all hosts on a specific network, the local address part of the IP address is set to 0. There are no user commands that use the broadcast capability, although such commands, or programs, can be developed.

Local Loopback Address

The Internet Protocol defines the special network address, 127.0.0.1, as a local loopback address. Local loopback is implemented in the kernel. The local loopback address must be set with the **ifconfig** command. This command is issued automatically in the **/etc/rc.include** file.

Sub-Network Addresses

The sub-network capability of the Interface Program makes it possible to divide a single network into multiple logical networks (***subnets***). For example, an organization can have a single Internet network address that is known to users outside the organization, yet configure its network internally into departmental subnets. Fewer Internet network addresses are required while local routing capabilities are enhanced.

As is explained under "IP Addresses" on page 1-29, a standard Internet address field has two parts, a network address and a local address. To make subnets possible, the local address part of an Internet address is divided into a subnet number (or ***mask***) and a host number, for example:

network_number subnet_number host_number

where:

network_number is Internet address for the network.
subnet_number is a field of a constant width for a given network.
host_number is a field that is at least one bit wide.

If the width of the *subnet_number* field is 0, the network is not organized into subnets, and addressing to the network is done with the Internet network address (*network_number*).

The **subnetmask** keyword must be set in the **/etc/net** file of a host if that host is to support subnets. Before the sub-network capability can be used, all hosts on the network must support it.

Figure 1-6 represents a class B network address with a 6-bit wide subnet field:

0 1	1 2 3 4 5 6 7 8 9 0 1 2 3 4 5	2 6 7 8 9 0 1	3 2 3 4 5 6 7 8 9 0 1
1 0	Network	Subnet	Host

Figure 1-6. Class B Address with Subnet

The bits that identify the subnet are specified by a bit mask and, therefore, are not required to be adjacent in the address. However, it is generally desirable for the subnet bits to be contiguous and located as the most significant bits of the local address.

Socket Addressing

The Interface Program provides a set of 16-bit port numbers within each host. Each host assigns port numbers independently. Therefore, it is possible, and in certain cases likely, for ports on different hosts to have port numbers that are not unique between hosts. To create an identifier that is unique throughout all Internet networks, the Interface Program concatenates the Internet address of the local host interface with the port number, producing an ***Internet socket address***.

Since the Internet address is always unique to a particular host on a network, so is the socket address for a particular socket on a particular host. Additionally, since each connection is fully specified by the pair of ports (sockets) it joins, every connection between Internet hosts is also uniquely identified.

The port numbers in the range 0-1023 are reserved for ports that are connected to processes that provide certain ***well-known services***. When a client process needs one of these well-known services at a particular host, the client process sends a ***service request*** to the socket address for the well-known port at the server host.

If a server process is listening at the well-known port, the server process then either services the request using the well-known port or it may transfer the connection to another port that is temporarily assigned for the duration of the connection to the client. This use of temporarily-assigned (or secondary) ports allows freeing the well-known port and allows the server to handle additional requests concurrently.

The port numbers up to 255 are reserved for official Internet services, and higher numbered ports are reserved for other well-known services that are common on Internet networks. The port numbers for well-known ports are listed in the **/etc/services** file. The port numbers above 1023 are generally used by processes that need a temporary port once an initial service request has been received. These port numbers are often generated randomly and are used on a first-come, first-served basis.

X.25 Addresses

All hosts on the X.25 network are assigned addresses, which can be obtained from the Network Information Center (NIC). (See NIC address on page vi for the mailing address.) Each address is an ASCII text string in what is known as ***host table format***. This section describes how these host table addresses are converted to X.25 addresses.

A host table address consists of the ASCII text string representations of four decimal numbers separated by periods, corresponding to the four octets of a 32-bit Internet address. The four decimal numbers are referred to in this sections as *n*, *h*, *l*, and *i*. Each host table address can be represented as *n.h.l.i*. Each of these four numbers has one, two, or three decimal digits and a value less than or equal to 255. For example, in the host table address `10.2.0.124`, *n*=10, *h*=2, *l*=0, and *i*=124. To convert a host table address to an X.25 address, the following rules are used:

1. If the *h* portion is less than 64, the host table address corresponds to the X.25 physical address:

   0000 0 ***iiihh*****00**

   where:

   0000 Required by the X.25 standard.

   0 Represents the physical address.

   iii Contains a 3-digit decimal representation of the *i* field, right-adjusted and padded with leading zeros if required.

   hh Contains a 2-digit decimal representation of the *h* field, right-adjusted and padded with leading zeros if required.

   00 Always set to 0.

   Using the previous example, the host table address `10.2.0.124` corresponds to X.25 physical address `000001240200`.

2. If the *h* portion is greater than or equal to 64, the host table address corresponds to the X.25 logical address:

   0000 1 ***rrrrr*****00**

   where:

   0000 Required by the X.25 standard.

   1 Represents the logical address.

rrrrr Contains a 5-digit decimal representation of the result R of the calculation:

```
R = h * 256 + i
```

Note that the decimal representation of R always requires 5 digits.

00 Always set to 0.

For example, the host table address 10.83.0.201 corresponds to the X.25 logical address 000012145500.

In both of the preceding cases, the n and l fields of the host table address are not used.

As a general X.25 note, remember that X.25 interfaces do not support server commands that require network broadcast. In AIX, these commands include **gated** for RIP routing, **routed**, and **rwhod**.

Packet Headers

The Interface Program implements standard local headers for IP and ARP packets sent over local area networks. These IP and ARP local headers are described in the following figures.

Baseband Adapter Local Headers

Figure 1-7 represents an IP or ARP local header for the Baseband Adapter:

Source Address	Destination Address	Type Field
6 bytes	6 bytes	2 bytes

Figure 1-7. Local Header, IP or ARP Packet, Baseband Adapter

The 2-byte ***type*** field in the Baseband Adapter local header distinguishes IP addresses from ARP addresses. The type numbers are:

Protocol	Type Number
IP	0800
ARP	0806

Figure 1-8. IP and ARP Type Numbers

Token-Ring Local Headers

Figure 1-9 represents an IP or ARP local header for the IBM Token-Ring Adapter or the 802.3 Adapter:

Medium Access Control (MAC) Header	Logical Link Control (LLC) Header

Figure 1-9. Local Header, IP or ARP Packet, IBM Token-Ring Adapter

The Medium Access Control (MAC) header for the IBM Token-Ring Adapter is composed of five fields, as Figure 1-10 shows.

MAC Header				
AC	FC	DA	SA	RI
1 byte	1 byte	6 bytes	6 bytes	≤ 18 bytes

Figure 1-10. Medium Access Control (MAC) Header, IBM Token-Ring Adapter Local Address

The MAC header fields are:

AC Access Control. The value in this field is x'00', which gives the header priority 0.

FC Field Control. The value in this field is x'40', which specifies the Logical Link Control frame.

DA Destination Address.

SA Source Address. If bit 0 of this field is set to 1, it indicates that routing information (RI) is present.

RI Routing Information. The RI fields are shown in Figure 1-11.

Routing Information (RI)	
RC	Segment Numbers (up to 8)
2 bytes	2 bytes each

Figure 1-11. MAC Header Routing Information, IBM Token-Ring Adapter Local Address

The RI fields are:

RC Routing Control. RC information is contained in bytes 0 and 1 of the RI field. The settings of the first two bits of the RC field have the following meanings:

- **bit (0) = 0** Use the non-broadcast route specified in the RI field.
- **bit (0) = 1** Create the RI field and broadcast to all rings.
- **bit (1) = 0** Broadcast through all bridges.
- **bit (1) = 1** Broadcast through limited bridges.

Segment Numbers Up to eight segment numbers of two bytes each to specify recipients of a limited broadcast.

802.3 Adapter

The Medium Access Control (MAC) header for the 802.3 Adapter is composed of three fields, as Figure 1-12 shows:

MAC Header		
DA	SA	Length
6 bytes	6 bytes	2 bytes

Figure 1-12. Medium Access Control (MAC) Header, 802.3 Adapter Local Address

The MAC header fields are:

DA Destination Address

SA Source Address.

Logical Link Control (LLC) Header

The Logical Link Control (LLC) header is composed of five fields, as Figure 1-13 shows.

LLC Header				
DSAP	SSAP	CONTROL	PROT_ID	TYPE
1 byte	1 byte	1 byte	3 bytes	2 bytes

Figure 1-13. Logical Link Control (LLC) Header, IBM Token-Ring Adapter Local Address

The fields of the LLC header are:

DSAP Destination Service Access Point. The value in this field is x'aa'.

SSAP Source Service Access Point. The value in this field is x'aa'.

CONTROL Determines the LLC commands and responses. There are three possible values for this field:

- **x'03'** Unnumbered Information (UI) frame. This is the normal, or unsequenced, way in which IBM Token-Ring Adapter data is transmitted through the network. TCP sequences the data.
- **x'AF'** Exchange Identification (XID) frame. This frame conveys the characteristics of the sending host.
- **x'E3'** Test frame. This frame supports testing of the transmission path, echoing back the data that is received.

PROT_ID Protocol ID. This field is reserved. It has a value of x'0'.

TYPE Specifies whether the packet is IP or ARP.

Assigned Numbers

For compatibility with the general network environment, well-known numbers are assigned for the Internet versions, networks, ports, protocols, and protocol options. Additionally, well-known names are also assigned to machines, networks, operating systems, protocols, services, and terminals. The Interface Program complies with the assigned numbers and names defined in RFC 1010, *Assigned Numbers*.

The Internet Protocol defines a 4-bit field in the IP header that identifies the version of the general internetwork protocol in use. For IP, this version number in decimal is 4. For details on the assigned numbers and names used by the Interface Program, refer to the **/etc/protocols** and **/etc/services** files included with the Interface Program. For further details on the assigned numbers and names, refer to RFC 1010 and the **/etc/services** file.

Security Features

This section describes the security features provided with the Interface Program, both in the standard mode and in controlled access mode, and discusses some security considerations that are appropriate in a network environment.

Overview of the Interface Program and Controlled Access Mode

Controlled access mode provides an additional level of security for those installations that require it. Certain Interface Program commands that are not trusted are disabled when controlled access mode is established, and restrictions are put in place to prohibit network communications with other hosts that are not operating at the same level of security as the local host. The commands that are not available in controlled access mode are:

rcp
rlogin
rsh and **remsh**
tftp and **utftp**
trpt
rlogind
rshd
tftpd.

The **securetcpip** command is used to convert a system from the standard level of security to the higher level known as controlled access mode. Once your system has been converted, you do not need to issue the **securetcpip** command again unless you re-install the Interface Program.

The following sections describe the security features of the Interface Program in terms of:

- Access control
- Auditing
- Trusted path, trusted shell, and the Secure Attention Key
- Hardcopy access and labeling
- Definition of the Network Trusted Computing Base.

Additional information on these features is included in the description of the appropriate command or function in later chapters of this book.

Access Control

The security policy for networking is an extension of the security policy for the AIX Operating System, and it consists of the following major components:

- User Authentication
- Connection Authentication
- Data Import and Export Security.

User authentication is provided at the remote host by a user name and password, the same as when a user logs in to the local system. Interface Program commands provided in controlled access mode, such as **ftp** and **telnet**, have the same requirements and go through the same verification process as commands in AIX.

In order to use the file transfer function available in controlled access mode, the **ftp** command requires two TCP connections, one for the protocol and one for data transfer. The protocol connection is primary and is secure because it is established on reliable communicating ports. The secondary connection is needed for the actual transfer of data, and both the local and remote host verify that the other end of this connection is established with the same host as the primary connection. If the primary and secondary connections are not established with the same host, **ftp** displays an error message stating that the data connection was not authenticated and then exits. This verification of the secondary connection prevents a third host from intercepting data intended for another host.

Security for data that is transferred into or out of the local host is provided by the IP security option. The IP security option, defined in RFC 1038, allows, but does not require, a level of security to be defined for the local host. In AIX, this security level is provided by the **netconfig** command and the **ip_security** keyword in the **/etc/net** file. A security level should be defined for each host operating in controlled access mode. Incoming data from hosts that have a security level defined must have a level equal to the one defined for the local host; otherwise, an error occurs and the data is not received.

Auditing

Network auditing is provided by the Interface Program, using the AIX auditing subsystem to audit both kernel network routines and application programs. The purpose of auditing is to record those actions that affect the security of the system and the person responsible for those actions.

The following types of events are audited:

Kernel Events

- Change configuration
- Change host ID
- Change route
- Connection
- Create socket
- Export object
- Import object

Application Events

- Access the network
- Change configuration
- Change host ID
- Change manually set route
- Configure mail
- Connection
- Export data
- Import data
- Set network time
- Write mail to a file

Creation and deletion of objects are audited by AIX. Application audit records suspend and resume auditing to avoid redundant auditing by the kernel.

For a list of the events audited by the Interface Program, see the **/etc/security/config** file. For information on configuring the auditing subsystem, see *Managing the AIX Operating System.* For information on the format of an audit trail record, see the **auditbin** system call in *AIX Operating System Technical Reference.*

Trusted Path, Trusted Shell, and Secure Attention Key (SAK)

The AIX Operating System provides the ***trusted path*** to prevent unauthorized programs from reading data from a user terminal. This path is used when a secure communication path with the system is required, such as when you are changing passwords or logging in to the system. AIX also provides the ***trusted shell***, **tsh**, which executes only trusted programs that have been tested and verified as secure. The Interface Program supports both of these features, along with the ***Secure Attention Key*** (SAK), which establishes the environment necessary for secure communication between you and the system. The local SAK is available whenever you are using an Interface Program program, and a remote SAK is available through the **telnet** command.

The local SAK has the same function in **telnet** that it has in other AIX application programs: it terminates the **telnet** process and all other processes associated with the terminal in which **telnet** was running. Inside the **telnet** program, however, you can send a request for a trusted path to the remote system using the **telnet send sak** command (while in **telnet** command mode). You can also define a single key to initiate the SAK request using the **telnet set sak** command. See "**telnet, tn**" on page 2-72 for more information on accessing a trusted path on a remote host.

Hardcopy Access and Labeling

Access to local hardcopy devices (printers, plotters, and so on) from remote hosts is controlled by the **/etc/hosts.equiv** and **/etc/hosts.lpd** files. The **hosts.equiv** file specifies foreign hosts that can execute commands, include remote printing, on the local host. The **hosts.lpd** file contains a list of hosts whose users have access to the local host *only* for printing. Users on any system that is not listed in the **hosts.lpd** or **hosts.equiv** file cannot print on local printers.

The labeling that appears on hardcopy printouts is defined on a system-wide basis by the system administrator. The remote printing facilities, **lprbe** and **lpd**, provide the same hardcopy labeling as defined for the system where the print command was issued. Details on how to set up this labeling are provided in the section on controlled access mode in *Managing the AIX Operating System* and in the description of the **/etc/security/config** file in *AIX Operating System Technical Reference.*

Definition of the Network Trusted Computing Base (NTCB)

The network contains both hardware and software mechanisms to implement the networking security features available through controlled access mode. This section defines the components of the Network Trusted Computing Base (NTCB).

The hardware security features for the network are provided by the network adapters used with TCP/IP. These adapters are programmed to control incoming data by receiving only data destined for the local system and broadcast data receivable by all systems.

The software component of the NTCB consists of only those programs that are considered trusted. (See the section on controlled access mode in *Managing the AIX Operating System* for a definition of trusted programs.) The programs and associated files that are part of a system operating in controlled access mode are listed in the following sections on a directory-by-directory basis.

/etc

Name	Owner	Group	Mode	Permissions
filters	root	system	0664	rw−rw−r−−
gated.conf	root	system	0664	rw−rw−r−−
gateways	root	system	0664	rw−rw−r−−
hosts	root	system	0664	rw−rw−r−−
hosts.equiv	root	system	0664	rw−rw−r−−
hosts.lpd	root	system	0664	rw−rw−r−−
ifconfig	root	system	4554	r−sr−xr−−

Name	Owner	Group	Mode	Permissions
inetd.conf	root	system	0664	rw – rw – r – –
named.boot	root	system	0664	rw – rw – r – –
named.data	root	system	0664	rw – rw – r – –
net	root	system	0664	rw – rw – r – –
netconfig	root	system	4554	r – sr – xr – –
networks	root	system	0664	rw – rw – r – –
protocols	root	system	0664	rw – rw – r – –
rc.sendmail	root	system	4554	r – sr – xr – –
rc.tcpip	root	system	0664	rw – rw – r – –
resolv.conf	root	system	0664	rw – rw – r – –
route	root	system	4554	r – sr – xr – –
securetcpip	root	system	4554	r – sr – xr – –
services	root	system	0664	rw – rw – r – –
syslogd	root	system	4554	r – sr – xr – –
3270.keys	root	system	0664	rw – rw – r – –
3270keys.rt	root	system	0664	rw – rw – r – –

/bin

Name	Owner	Group	Mode	Permissions
hostname	root	bin	0555	r – xr – xr – x
netstat	root	bin	0555	r – xr – xr – x

/usr/adm/sendmail

Name	Owner	Group	Mode	Permissions
aliases	root	system	660	rw – rw – – – –
sendmail.cf	root	system	660	rw – rw – – – –
sendmail.hf	root	system	660	rw – rw – – – –
smdemon.cleanu	root	system	660	rw – rw – – – –

/usr/bin

Name	Owner	Group	Mode	Permissions
arp	root	bin	4555	r−sr−xr−x
finger	root	bin	4555	r−sr−xr−x
ftp	root	bin	4555	r−sr−xr−x
host	root	bin	4555	r−sr−xr−x
hostid	root	bin	4555	r−sr−xr−x
lprbe	root	bin	4555	r−sr−xr−x
ping	root	bin	4555	r−sr−xr−x
rexec	root	bin	4555	r−sr−xr−x
ruptime	root	bin	4555	r−sr−xr−x
rwho	root	bin	4555	r−sr−xr−x
setclock	root	bin	4555	r−sr−xr−x
talk	root	bin	4555	r−sr−xr−x
telnet	root	bin	4555	r−sr−xr−x
tn	root	bin	4555	r−sr−xr−x
trpt	root	bin	4555	r−sr−xr−x
xftp	root	bin	4555	r−sr−xr−x

/usr/lib

Name	Owner	Group	Mode	Permissions
sendmail	root	system	6551	r−sr−s−−x
sendmail.sh	root	system	6554	r−sr−sr−x

/usr/lpp/tcpip/etc

Name	Owner	Group	Mode	Permissions
fingerd	root	system	4554	r−sr−xr−−
ftpd	root	system	4554	r−sr−xr−−
gated	root	system	4554	r−sr−xr−−
inetd	root	system	4554	r−sr−xr−−

Name	Owner	Group	Mode	Permissions
lpd	root	system	4554	r−sr−xr−−
named	root	system	4554	r−sr−xr−−
rexecd	root	system	4554	r−sr−xr−−
routed	root	system	4554	r−sr−xr−−
rwhod	root	system	4554	r−sr−xr−−
talkd	root	system	4554	r−sr−xr−−
telnetd	root	system	4554	r−sr−xr−−

/usr/lpp/tcpip/samples

Name	Owner	Group	Mode	Permissions
addrs.awk	root	system	0774	rwxrwx−−x
hosts.awk	root	system	0774	rwxrwx−−x

/usr/spool/lpd

Name	Owner	Group	Mode	Permissions
lpd (directory)	root	system	0755	rwxr−xr−x

/usr/spool/rwho

Name	Owner	Group	Mode	Permissions
rwho (directory)	root	system	0755	rwxr−xr−x

Interface Program Command Security

The **ftp**, **lprbe**, **rexec**, **securetcpip**, and **telnet** commands provide the following forms of system and data security:

ftp or **xftp**
: The **ftp** function provides a secure environment for transferring files. When a user invokes **ftp** to a foreign host, the user is prompted for a login ID. A default login ID is shown—the user's current login ID on the local host. If a password is associated with that login ID at the foreign host, the user is prompted for a password.

  The automatic login process searches the **$HOME/.netrc** file for the user's ID and password to use at the foreign host. For security, the permissions on **$HOME/.netrc** must be set to 600 (read and write by owner only). Otherwise, automatic login fails.

  Note: Since use of **.netrc** requires storage of passwords in a non-encrypted file, the automatic login feature of **ftp** is not available when your system is operating in controlled access mode.

  For more information, see "**ftp**, **xftp**" on page 2-9 and "**.netrc**" on page 4-68.

lprbe
: The **/etc/hosts.equiv** and **/etc/hosts.lpd** files provide a secure environment for the **lprbe** (remote print) command. The only foreign hosts that can use a particular host as a print server are the ones listed in the **/etc/hosts.equiv** or **/etc/hosts.lpd** file on the print server host. For more information about **/etc/hosts.equiv** and **/etc/hosts.lpd**, see "**hosts.equiv**" on page 4-28 and "**hosts.lpd**" on page 4-30.

rexec
: The **rexec** command provides a secure environment for executing commands on a foreign host. The user is prompted for both a login ID and a password.

  An automatic login feature causes **rexec** to search the **$HOME/.netrc** file for the user's ID and password on a foreign host. For security, the permissions on **$HOME/.netrc** must be set to 600 (read and write by owner only). Otherwise, automatic login fails.

  Note: Since use of **.netrc** requires storage of passwords in a non-encrypted file, the automatic login feature of **rexec** is not available when your system is operating in controlled access mode.

  For more information, see "**rexec**" on page 2-51 and "**.netrc**" on page 4-68.

securetcpip
: The **securetcpip** command enables controlled access mode, the AIX networking security features. Access to commands that are not trusted is removed from the system when this command is issued, and the IP security option is defined for all of the interfaces listed in the **/etc/net**

file. For more information, see "**securetcpip**" on page 2-67 and "**net**" on page 4-52.

telnet or **tn** The **telnet** (TELNET) function provides a secure environment for login to a foreign host. The user is prompted for both a login ID and a password. The user's terminal is treated just like a terminal connected directly to the host. That is, access to the terminal is controlled by permission bits; other users (group and other) do not have read access to the terminal, but they can write messages to it if the owner gives them write permission. In controlled access mode, **telnet** also provides access to a trusted shell on the remote system through the Secure Attention Key (SAK). This key sequence differs from the sequence that invokes the local trusted path and can be defined within **telnet**.

For more information and an example, see "**telnet**, **tn**" on page 2-72.

Data Security and Information Protection

The Interface Program for use with TCP/IP does not encrypt user data transmitted through the network. Therefore, it is suggested that users identify any risk in communication that could result in the disclosure of passwords and other sensitive information, and based on that risk, apply appropriate countermeasures.

The use of this product in a Department of Defense (DoD) environment may require adherence to DoD 5200.5 and NCSD - 11 for communications security.

The Interface Program provides the IP security option in the **/etc/net** file to comply with the DoD Basic Security section of RFC 1038, *Revised IP Security Option.* Additional information, including addresses, for the accrediting authorities whose protection rules apply at each level of security is available in RFC 1038.

Program Customization

This section describes the usual means for customizing the Interface Program to meet your requirements.

EMULATE
The **telnet** command uses an environment variable called **EMULATE** that defines the emulation mode used by TELNET. For more information, see "**telnet**, **tn**" on page 2-72.

TNESC
The **telnet** command uses an environment variable called **TNESC** that defines the escape character used by TELNET. For more information, see "**telnet**, **tn**" on page 2-72.

/etc/filters
The **filters** file specifies the filters available on a particular host when it acts as a print server. For more information, see "**filters**" on page 4-3.

/etc/ftpusers
The **ftpusers** file specifies local user names that cannot be used by remote FTP clients. For more information, see "**ftpusers**" on page 4-5.

/etc/gated.conf
The **gated.conf** file contains configuration information for the **gated** daemon. For more information, see "**gated**" on page 3-9 and "**gated.conf**" on page 4-7.

/etc/gateways
The **gateways** file identifies gateways for the **routed** command. For more information, see "**routed**" on page 3-28 and "**gateways**" on page 4-21.

/etc/hosts
The **/etc/hosts** file contains the names and associated addresses for hosts on the network. That is, **/etc/hosts** lists the foreign hosts with which a local host can communicate. For more information, see "**hosts**" on page 4-24.

/etc/hosts.equiv
The **/etc/hosts.equiv** file contains the names of the foreign hosts that are permitted to perform remote operations on a particular host. For more information, see "**hosts.equiv**" on page 4-28 and "**lpd**" on page 3-16.

/etc/hosts.lpd
The **/etc/hosts.lpd** file contains the names of the foreign hosts that are permitted to perform remote print operations to a particular print server host. For more information, see "**hosts.lpd**" on page 4-30 and "**lpd**" on page 3-16.

/etc/inetd.conf

The **inetd.conf** file is the default configuration file for the **inetd** server, which defines how **inetd** handles Internet service requests. For more information, see "**inetd**" on page 3-13 and "**inetd.conf**" on page 4-32.

/etc/named.boot

The **named.boot** file is the default configuration (or boot) file for the **named** server, which defines how **named** initializes the DOMAIN nameserver database. For more information, see "**named**" on page 3-20 and "**named.boot**" on page 4-34.

/etc/named.*

The **named.*** files define the data that **named** uses to initialize the DOMAIN nameserver database. For more information, see "**named**" on page 3-20 and "**named.***" on page 4-37.

/etc/net

The **/etc/net** file contains a description of the interfaces available on a particular host. For more information, see "**net**" on page 4-52.

/etc/networks

The **/etc/networks** file contains information about known networks. For more information, see "**networks**" on page 4-58.

/etc/passwd

The **finger** command provides information about a specific user. This information is retrieved from the `full name` and `site information` fields of the **/etc/passwd** file. The format of the `full name` is the user's real name (or whatever is entered in the `full name` field of **/etc/passwd**), not the user's user ID. The `site information` field is 20 characters long and has the following format (fields separated by commas, no imbedded blanks):

```
office_number,office_phone_extension,home_phone
```

The `office_phone_extension` field can be up to 4 characters long. The `home_phone` field can be either 7 or 10 characters long.

Note: To update the **/etc/passwd** file, use the **adduser** command.

If you want to send additional information to other users who run **finger** on your user ID, you can include the following files in your home directory:

.plan A file that contains plans. The **.plan** file can contain more than one line.

.project A file that states what project you are currently working on. The **.project** file can contain only one line.

For additional information about the **finger** command, see "**finger**" on page 2-6.

/etc/rc.tcpip
The **/etc/rc.tcpip** file is a shell script that defines which adapters are used for the Interface Program, initializes the host names and routing tables, and starts the Interface Program daemons. The **/etc/rc.tcpip** file may be tailored to the requirements of a particular host. For more information, see "**rc.tcpip**" on page 4-62 and Appendix A, "Customizing the Program."

/etc/resolv.conf
The **resolv.conf** file defines information on the DOMAIN name server for the local resolver routines. For more information, see "**resolv.conf**" on page 4-64.

/etc/3270.keys
The **/etc/3270.keys** file contains the default key mapping for use with TELNET (3270).

$HOME/.netrc
The **$HOME/.netrc** file contains the user ID and password information required by the automatic login features of **rexec** and **ftp**. For more information, see "**.netrc**" on page 4-68.

$HOME/.rhosts
The **$HOME/.rhosts** file contains a list of remote users who are not required to supply a login password when they execute the **rlogin** and **rsh** commands using a local user account. For more information, see "**.rhosts**" on page 4-70.

$HOME/.3270keys
The **$HOME/.3270keys** file contains a user-specific keyboard mapping for TELNET (3270). For more information, see "**.3270keys**" on page 4-72.

Chapter 2. User Commands

CONTENTS

About This Chapter

This chapter describes the user commands, or application programs, included with the IBM RT Interface Program for use with TCP/IP. Chapter 3, "Server Commands" on page 3-1 describes the Interface Program server commands, or daemons.

The following is a list of the user commands, grouped according to function:

- Remote file transfers

  ftp or **xftp**
  rcp
  tftp
  utftp
  uucp[1]
  uuto[1]

- Remote mail and conversations

  Message Handling
  (MH) commands[1]
  mail[1]
  talk

- Remote login, command execution, and printing

  lprbe
  rexec
  rlogin
  rsh or **remsh**
  telnet or **tn**
  uux[1]

- Network management

  arp
  finger
  host
  hostid
  hostname
  ifconfig
  netconfig
  netstat
  ping
  route
  ruptime
  rwho
  setclock
  securetcpip
  trpt

In this chapter, the user commands are organized alphabetically.

---

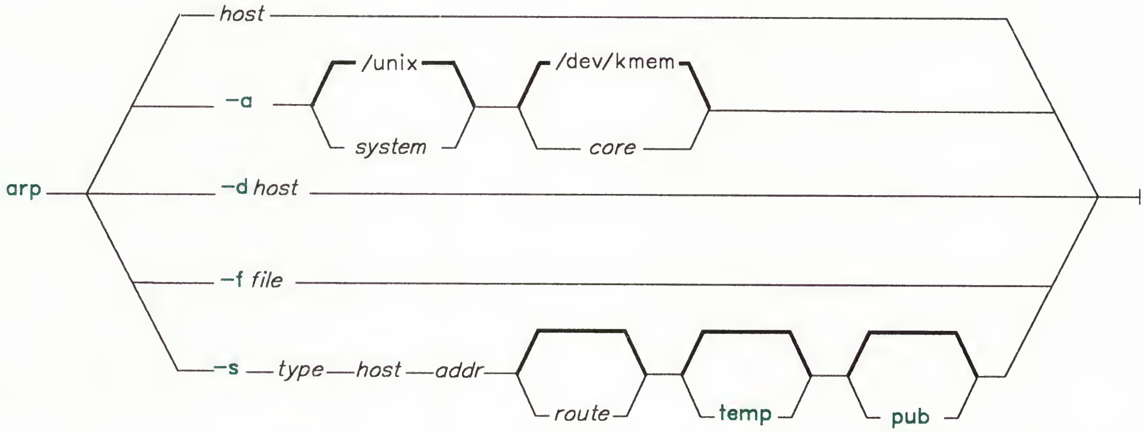
1 These commands are not part of the Interface Program. For a description, refer to *AIX Operating System Commands Reference.*

arp

Purpose

Displays or modifies the Internet to hardware address translation tables.

Syntax



A5ACC044

Description

The **arp** command displays and modifies the Internet address to hardware address translation tables that are used by the Address Resolution Protocol. The ARP tables can be displayed by any user, but can only be modified by a user with superuser authority.

Flags

-a [*system***] [***core***]** Displays all of the current ARP entries by reading the table from the file *core* (default **/dev/kmem**) based on the kernel file *system* (default **/unix**).

-d *host* Deletes map table entry for the specified *host* if the user issuing the command has superuser authority.

-f *file*
: Reads entries from the specified *file* and adds those entries to the ARP tables. Entries in the file are in the following form:

type host address [*route*] [**temp**] [**pub**]

where:

type
: Defines the type of adapter interface from the following list:

ether
: Specifies an IBM Baseband Adapter.

802.3
: Specifies a customer-supplied 802.3 Adapter.

802.5
: Specifies an IBM Token-Ring Adapter.

host
: Specifies the remote host identified by this entry.

address
: Contains the hardware address of the adapter for this host as six hexadecimal bytes, separated by colons. The **netstat -v** command displays the hardware address for the local system.

route
: Defines the route for a Token-Ring interface, as contained in the Token-Ring header.

temp
: Specifies that this ARP table entry is temporary. When this argument is not used, the table entry is permanent.

pub
: Indicates the table entry will be *published*, and this system will act as an ARP server, responding to requests for *host* even though the host address is not its own.

-s *type host address route* **[temp] [pub]**
: Creates a single ARP entry for the specified *host*, with the same arguments as explained in the preceding discussion of the **-f** flag.

Examples

1. To display the Internet address to hardware address mapping tables used by the Address Resolution Protocol for the local host, which has only one interface defined:

```
$ arp -a
ethernet ARP table:
  host4 (192.9.201.18) at 0:dd:0:39:af:0
$ _
```

2. To add a single entry to the ARP mapping tables only until the next time the system is restarted:

   ```
   $ arp -s 802.3 host2 0:dd:0:a:85:0 temp
   ```

   Note: This command requires superuser authority.

3. To add multiple entries to the ARP mapping tables from the file **newentries**:

   ```
   $ arp -f newentries
   $ _
   ```

   Note: This command requires superuser authority.

Related Information

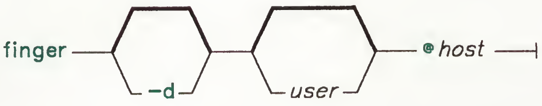
In this book: "Address Resolution Protocol (ARP)" on page 1-15.

finger

Purpose

Shows user information.

Syntax



A5ACC002

Description

The **finger** command displays information about the users currently using the specified host. The host you specify must have a **fingerd** server started by its **inetd** server and have an account defined that limits the access of **fingerd**. For more information on setting up this account for your host, see "**fingerd**" on page 3-3.

The information **finger** provides includes:

- *login name*
- *terminal name* (an asterisk (*) indicates that write permission is denied)
- *login time.*

The times shown are in minutes if only an integer is shown, in hours and minutes if a colon (:) is shown, and in days and hours if a d is shown.

When you specify users with the *user* parameter, you can specify either the user's first name, last name, or account name. When you specify users, **finger** at the specified host returns information about those users only. Additionally, **finger** uses a longer format that also includes:

- *full name*
- *idle time*
- *office_location*, *office_phone_number*, and *home_phone* (if known)
- The contents of a **.plan** file in the user's home directory, if one exists
- The contents of a **.project** file in the user's home directory, if one exists.

If you want to send additional information to other users who run **finger** on your user ID, you can include the following files in your home directory:

.plan A file that contains plans. The **.plan** file can contain more than one line.

.project A file that states what project you are currently working on. The **.project** file can contain only one line.

For other information about **finger**, see "Program Customization" on page 1-48.

Flag

-d Starts socket-level debugging.

Examples

1. To get information about all users logged in to the host `host1`:

```
$ finger @host1
   smith    console  Mar 15 13:19
   green    pts0     Mar 15 13:01
   thomas   tty0     Mar 15 13:01
$ _
```

   User `smith` is logged in at the console, user `green` is logged in from a pseudo teletype line, and user `thomas` is logged in from a tty.

2. To get information about user `smith` at `host1`:

```
$ finger smith@host1
Login name:  smith        In real life:  Sam T. Smith
Office:  3D08 ext5555    Home phone:  987-6543
Directory:  /u/smith     Shell:  /bin/sh
Not logged in.
No plan.
$ _
```

3. To get information about user `thomas` at `host1` (`thomas` has both a **.plan** and a **.project** file in his home directory):

```
$ finger thomas@host1
Login name:  thomas     In real life:  A. B. Thomas
Office:  3D10  ext5322  Home phone:  210-9876
Directory:  /u/thomas   Shell:  /bin/sh
On since April 16 11:06:18 on console  1 minute 28 seconds Idle Time
Project:  aquatic entomology
Plan:
Complete Phase 1 research by end of second quarter.
Produce draft report by end of year.
$ _
```

Files

/etc/utmp	Defines who is currently logged in.
/etc/passwd	Defines user accounts, names, home directories, and so on.
/etc/security/passwd	Defines user passwords.
$HOME/.plan	Optional file that contains a one-line description of a user's plan.
$HOME/.project	Optional file that contains a user's project assignment.

Related Information

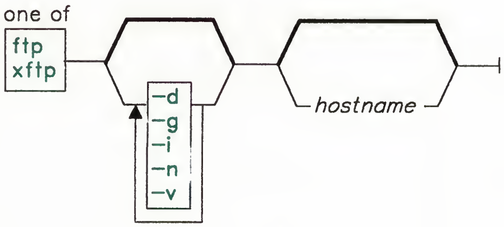
In this book: "**rwho**" on page 2-65 and "**fingerd**" on page 3-3.

ftp, xftp

Purpose

Transfers files between a local and a remote host.

Syntax



A5ACC018

Description

The **ftp** command is the interface to the File Transfer Protocol (FTP). This command uses FTP to transfer files between the local host and a remote host or between two remote hosts. The **xftp** command performs the same function as the **ftp** command. In the remainder of this section, the name **ftp** refers to either the **ftp** or **xftp** command.

The FTP protocol is designed to allow transferring data between hosts that use dissimilar file systems. Therefore, although the protocol provides a lot of flexibility for transferring data, it does not attempt to preserve file attributes (such as the protection mode or modification times of a file) that are specific to a particular file system. Additionally, the FTP protocol makes few assumptions about the overall structure of a file system and does not provide or allow such things as recursively copying subdirectories.

Note: If you are transferring files between AIX systems and need to preserve file attributes or need to recursively copy subdirectories, you can use the **rcp** command.

If you execute **ftp** and do not specify *hostname*, **ftp** immediately displays the `ftp>` prompt and waits for an **ftp** command. To connect to a foreign host, you then execute the **open** command. (The commands that **ftp** recognizes are discussed in "Subcommands" on page 2-11.) When **ftp** connects to the foreign host, **ftp** then prompts for the login name and password before displaying the `ftp>` prompt again. The **ftp** command fails if no password is defined at the remote host for the user name you are trying to log in as.

If you do specify the *hostname* of a foreign host, **ftp** tries to immediately establish a connection to the specified host. If **ftp** connects successfully, **ftp** searches for a local **.netrc** file in your current directory or home directory. If the file exists, **ftp** searches the file for an entry that initiates the login process and command macro definitions for the foreign host. If the **.netrc** file or auto-login entry does not exist, or if your system is operating in controlled access mode, **ftp** displays the `ftp>` prompt and waits for a command.

If **ftp** finds a **.netrc** auto-login entry for the specified host, **ftp** attempts to use the information in that entry to automatically log in to the foreign host. **ftp** also loads any command macros defined in the entry. In some cases (for example, when the required password is not listed in an auto-login entry), **ftp** prompts for the password before displaying the `ftp>` prompt. Once **ftp** completes the auto-login process, **ftp** executes the `init` macro if the macro is defined in the auto-login entry. If the `init` macro does not exist or does not contain a **quit** or **bye** command, **ftp** then displays the `ftp>` prompt and waits for a command.

Note: The remote user name that you specify either at the prompt or in a **.netrc** file must exist and have a password defined at the remote host, or **ftp** fails.

The **ftp** command interpreter, which handles all commands entered at the `ftp>` prompt, provides facilities that are not available with most file transfer programs, such as: the handling of filename parameters to **ftp** commands, the ability to collect a group of commands into a single command macro, and the ability to load macros from a **.netrc** file. These facilities are designed to allow simplifying repeated tasks and to allow using **ftp** in unattended mode.

The command interpreter handles file name parameters according to the following rules:

1. If **-** is specified for the parameter, standard input is used for read operations and standard output is used for write operations.
2. If the preceding check does not apply and filename expansion is enabled (see the **glob** subcommand), the interpreter expands the filename according to the rules of the C shell. However, if **ftp** expects a single filename (such as for the **get** and **put** subcommands), **ftp** uses only the first name generated.
3. For the **get**, **put**, **mget**, and **mput** subcommands, the interpreter has the ability to translate and map between different local and remote filename syntax styles (see the **case**, **ntrans**, and **nmap** subcommands) and the ability to modify a local filename if it is not unique (see the **runique** subcommand). Additionally, **ftp** can send instructions to a remote **ftpd** server to modify a remote filename if it is not unique (see the **sunique** subcommand).

Note: The **ftp** command interpreter does not support pipes.

To end an **ftp** session when you are running interactively, use the **bye** or **quit** subcommand or the **END OF FILE** key at the `ftp>` prompt. To end a file transfer before it has completed, use the INTERRUPT key. Sending transfers (those from the local host to

the foreign host) are normally halted immediately. Receiving transfers are halted by sending an FTP ABOR instruction to the remote FTP server and discarding all incoming file transfer packets until the remote server stops sending them. If the remote server does not support the ABOR instruction, the `ftp>` prompt will not appear until the remote server has sent all of the requested file. Additionally, if the remote server does something unexpected, the local **ftp** process may need to be ended manually.

Flags

The following flags can be entered either on the shell command line or at the `ftp>` prompt:

-d Enables debugging by turning on the logging feature. See the **debug** subcommand.

-g Disables the expansion of metacharacters in file names. Interpreting metacharacters may be referred to as expanding (sometimes called ***globbing***) a file name. See the **glob** subcommand.

-i Turns off interactive prompting during multiple file transfers. See the **prompt**, **mget**, **mput**, and **mdelete** subcommands for descriptions of prompting during multiple file transfers.

-n Prevents an automatic login on the initial connection. Otherwise, **ftp** searches for a **.netrc** entry that describes the login and initialization process for the remote host. See the **user** subcommand.

-v Displays all the responses from the remote server and provides data transfer statistics. This is the default display mode. See the **verbose** subcommand.

Subcommands

The following **ftp** subcommands can be entered at the `ftp>` prompt. If a parameter for a subcommand includes blank characters, the parameter can be quoted using double-quote (") characters.

![*command* [*parameters*]]
Invokes an interactive shell on the local host. An optional command, with one or more optional parameters, can be given with the shell command.

$ *macroname* [*parameters*]
Executes the specified macro, previously defined with the **macdef** command. Parameters are not expanded. See the **macdef** subcommand for further information.

? [*subcommand*] Displays a message describing the subcommand. If you do not specify *subcommand*, **ftp** displays a list of known subcommands.

account [*password*]
Sends a supplemental password that a remote host other than an RT system may require before granting access to its resources. If the password is not supplied with the command, the user is prompted for the password. The password does not appear on the screen.

append *localfile* [*remotefile*]
Appends a local file to a file on the remote host. If the remote file name is not specified, the local file name is used, altered by any setting made with the **ntrans** or **nmap** subcommand. **append** uses the current values for **form**, **mode**, **struct**, and **type** while appending the file. For more information on each of these subcommands, see their individual descriptions, which follow.

ascii Sets the file transfer type to network ASCII. This is the default. File transfer may be more efficient with binary-image transfer. Refer to the **binary** subcommand description.

bell Sounds a bell after the completion of each file transfer.

binary Sets the file transfer type to binary image. This can be more efficient than an ASCII transfer.

bye Ends the File Transfer session and exits **ftp**. Same as **quit**.

case Sets a toggle for the case of file names. When **case** is on, remote file names that appear in all capital letters are changed from uppercase to lowercase when written in the local directory. The default is off (uppercase remote file names are written in uppercase in the local directory).

cd *remotedirectory* Changes the working directory on the remote host to the specified directory.

cdup Changes the working directory on the remote host to the parent of the current directory.

close Ends the File Transfer session, but does not exit **ftp**. Defined macros are erased. Same as **disconnect**.

cr Strips the carriage return character from a carriage return/linefeed sequence when receiving records during ASCII-type file transfers. (**ftp** terminates each ASCII-type record with a carriage return/linefeed during file transfers.) This conforms with the AIX single linefeed record delimiter. Records on non-AIX remote hosts may have single linefeeds imbedded in records. To distinguish these imbedded linefeeds from record delimiters, set **cr** to off. **cr** toggles between on and off.

delete *remotefile* Deletes the specified remote file.

debug [on | off] Prints each command sent to the remote host, preceded by the string **-->** when **debug on** is specified. Use the **debug off** subcommand to stop the debug record keeping.

dir [*remotedirectory*][*localfile*]
Writes a listing of the contents of the *remotedirectory* to the file *localfile*. If *directory* is not specified, **dir** lists the contents of the current remote directory. If *localfile* is not specified or is **-**, **dir** displays the listing on the local terminal.

disconnect See **close**.

form *format* Specifies the form of the file transfer. The only form available is `file`.

get *remotefile* [*localfile*]
Copies the remote file to the local host. If *localfile* is not specified, the remote file name is used locally and is altered by any settings made by the **case**, **ntrans**, and **nmap** subcommands. **ftp** uses the current settings for **type**, **form**, **mode**, and **struct** while transferring the file. For additional information, refer to the description of each of these subcommands.

glob Toggles file name expansion (globbing) for **mdelete**, **mget**, and **mput**. If globbing is off, file name parameters for these subcommands are not expanded. Globbing for **mput** is done locally in the same way as for the **csh** command. For **mdelete** and **mget**, each file name is expanded separately at the remote machine and the lists are not merged. The expansion of a directory name may be different than the expansion of a file name, depending on the remote host and the **ftp** server. To preview the expansion of a directory name, use the **mls** subcommand:

`mls` *remotefilename* -

To transfer an entire directory subtree of files, transfer a **tar** archive of the subtree in binary form, rather than using **mget** or **mput**.

hash Toggles hash sign (#) printing. When **hash** is on, **ftp** displays one hash sign for each data block (1024 bytes) transferred.

help [*subcommand*]
Displays help information. Refer to the **?** subcommand.

lcd [*directory*] Changes the working directory on the local host. If you do not specify a directory, **ftp** uses your home directory.

ls [*remotedirectory*] [*localfile*]
Writes an abbreviated file listing of a remote directory to a local file. If *remotedirectory* is not specified, **ftp** lists the current remote

directory. If *localfile* is not specified or is **-**, **ftp** displays the listing on the local terminal.

macdef *macroname*
Defines a subcommand macro. Subsequent lines up to a null line (two consecutive linefeeds) are saved as the text of the macro. Up to 16 macros containing at most 4096 characters for all macros can be defined. Macros remain defined until redefined or a **close** is executed.

$ and \ are special characters in **ftp** macros. A $ followed by one or more numbers is replaced by the corresponding macro parameter on the invocation line (refer to the **$** subcommand). A $ followed by an `i` indicates that the macro is to loop, with `$i` being replaced by consecutive parameters on each pass. The first macro parameter is used on the first pass, the second parameter is used on the second pass, and so on. A \ prevents special treatment of the next character. Use the \ to turn off the special meanings of $ and \. For a description of the **macdef** format, see "**.netrc**" on page 4-68.

mdelete *remotefiles*
Expands *remotefiles* at the foreign host and deletes the indicated remote files.

mdir [*remotedirectories localfile*]
Expands *remotedirectories* at the foreign host and writes a listing of the contents of the *remotedirectories* to the *localfile*. If the *remotedirectories* parameter contains a pattern-matching character, **mdir** prompts for a *localfile* if none is specified. If the *remotedirectories* parameter is a list of remote directories, separated by blanks, the last argument in the list must be either a local file name or a -. If *localfile* is **-**, **mdir** displays the listing on the local terminal. If interactive prompting is on (refer to the **prompt** subcommand), **ftp** prompts the user to verify that the last parameter is a local file and not a remote directory.

mget *remotefiles*
Expands *remotefiles* at the foreign host and copies the indicated remote files to the current directory on the local host. Refer to the **glob** subcommand for more information on file name expansion. The remote file names are used locally and are altered by any settings made by the **case**, **ntrans**, and **nmap** subcommands. **ftp** uses the current settings for **type**, **form**, **mode**, and **structure** while transferring the files. Refer to the description of each of these subcommands for additional information.

mkdir [*remotedirectory*]
Creates the directory *remotedirectory* on the foreign host.

mls [*remotedirectories localfile*]
Expands *remotedirectories* at the foreign host and writes an abbreviated file listing of the indicated remote directories to a local file. If the *remotedirectories* parameter contains a pattern-matching character, **mls** prompts for a *localfile* if none is specified. If the *remotedirectories* parameter is a list of remote directories, separated by blanks, the last argument in the list must be either a local file name or a **-**. If *localfile* is **-**, **mls** displays the listing on the local terminal. If interactive prompting is on (refer to the **prompt** subcommand), **ftp** prompts the user to verify that the last parameter is a local file and not a remote directory.

mode [*modename*] Sets file transfer mode. The only mode available is **stream**.

mput [*localfiles*] Expands *localfiles* at the local host and copies the indicated local files to the foreign host. Refer to the **glob** subcommand for more information on file name expansion. The local file names are used at the foreign host and are altered by any settings made by the **ntrans** and **nmap** subcommands. **ftp** uses the current settings for **type**, **form**, **mode**, and **structure** while transferring the files. Refer to the description of each subcommand for additional information.

nmap [*inpattern outpattern*]
Sets or unsets the file name mapping mechanism. If no parameters are specified, file name mapping is turned off. If parameters are specified, source file names are mapped for **mget** and **mput** operations and for **get** and **put** operations when the destination file name is not specified. This subcommand is useful when the local and foreign hosts use different file-naming conventions or practices. Mapping follows the pattern set by *inpattern* and *outpattern*.

inpattern is the template for incoming file names, which may have already been processed according to the **case** and **ntrans** settings. The template variables `$1` through `$9` can be included in *inpattern*. All characters in *inpattern* other than `$` and protected `$`s (that is, `\$`) define the values of the template variables. For example, if the *inpattern* is `$1.$2` and the remote file name is `mydata.dat`, the value of `$1` is `mydata` and the value of `$2` is `dat`.

outpattern determines the resulting file name. The variables `$1` through `$9` are replaced by their values as derived from *inpattern* and the variable `$0` is replaced by the original file name. Additionally, the sequence `[`*seq1*`,`*seq2*`]` is replaced by the value of *seq1* if *seq1* is not null; otherwise, it is replaced by the value of *seq2*. For example, the subcommand:

```
nmap $1.$2.$3 [$1,$2].[$2,file]
```

would yield `myfile.data` from `myfile.data` or `myfile.data.old`, `myfile.file` from `myfile`, and `myfile.myfile` from `.myfile`. Spaces can be included in *outpattern*, as in: `nmap |sed "s/*$//" > $1`. Use the backslash (\) character to prevent the special meanings of $, [,], and , in *outpattern*.

ntrans [*inchars* [*outchars*]]
Sets or unsets the file name character translation mechanism. If no parameters are specified, character translation is turned off. If parameters are specified, characters in source file names are translated for *mget* and *mput* operations and for *get* and *put* operations when the destination file name is not specified. This subcommand is useful when the local and foreign hosts use different file naming conventions or practices. Character translation follows the pattern set by *inchars* and *outchars*. Characters in a source file name matching characters in *inchars* are replaced by the corresponding characters in *outchars*. If the string *inchars* is longer than the string *outchars*, characters in *inchars* are deleted if they have no corresponding character in *outchars*.

open *host* [*port*]
Establishes a connection to the FTP server at the specified *host*. If the optional port number is specified, **ftp** will attempt to connect to a server at that port. If the auto-login feature is set (that is, `-n` was not specified on the command line), **ftp** will attempt to automatically log the user in to the FTP server.

prompt
Toggles interactive prompting. If interactive prompting is on (the default), **ftp** will prompt for verification before retrieving, sending, or deleting multiple files during **mget**, **mput**, and **mdelete** operations. Otherwise, **ftp** will act accordingly on all files specified.

proxy [*subcommand*]
Executes an **ftp** command on a secondary control connection. This subcommand allows **ftp** to simultaneously connect to two remote FTP servers for transferring files between the two servers. The first **proxy** subcommand should be an **open** to establish the secondary control connection. Enter the subcommand `proxy ?` to see the other **ftp** subcommands that are executable on the secondary connection. The following subcommands behave differently when prefaced by **proxy**:

- **open** will not define new macros during the auto-login process.
- **close** will not erase existing macro definitions.
- **get** and **mget** transfer files from the host on the primary connection to the host on the secondary connection.
- **put**, **mput**, and **append** transfer files from the host on the secondary connection to the host on the primary connection.

File transfers require that the FTP server on the secondary connection must support the PASV (passive) instruction.

put *localfile* [*remotefile*]
Stores a local file on the remote host. If you do not specify *remotefile*, **ftp** uses the local file name to name the remote file, and the remote file name is altered by any settings made by the **ntrans** and **nmap** subcommands. **ftp** uses the current settings for **type**, **form**, **mode**, and **structure** while transferring the files. Refer to the description of each subcommand for additional information.

pwd Displays the name of the current directory on the foreign host.

quit Ends the file transfer session and exits **ftp**. A synonym for **bye**.

quote *string* Sends the specified *string* verbatim to the foreign host. Quoting commands that involve data transfers can produce unpredictable results.

recv *remotefile* [*localfile*]
Copies the remote file to the local host. A synonym for **get**.

remotehelp [*subcommand*]
Requests help from the remote FTP server.

rename *fromname toname*
Renames a file on the foreign host.

reset Clears the reply queue. This command resynchronizes the command/reply sequencing between the local **ftp** process and the remote FTP server. **reset** can be used to resynchronize when a remote server violates the FTP command/reply rules.

rmdir *remotedirectory*
Removes the directory *remotedirectory* at the foreign host.

runique Toggles creating unique file names for local destination files during **get** and **mget** operations. If unique local file names is off (the default), **ftp** overwrites local files. Otherwise, if a local file has the same name as specified for a local destination file, **ftp** modifies the specified name of the local destination file with `.1`. If a local file is already using the new name, **ftp** appends the postfix `.2` to the specified name. If a local file is already using this second name, **ftp** continues incrementing the postfix until it either finds a unique file name or reaches `.99` without finding a unique name. If **ftp** cannot find a unique name, **ftp** reports an error and the transfer does not take place. Note that **runique** does not affect local file names generated from a shell command.

send *localfile* [*remotefile*[
Stores a local file on the remote host. A synonym for **put**.

sendport
Toggles the use of FTP PORT instructions. By default, **ftp** uses a PORT instruction when establishing a connection for each data transfer. When the use of PORT instructions is disabled, **ftp** does not use PORT instructions for data transfers. The PORT instruction is useful when dealing with FTP servers that ignore PORT instructions while incorrectly indicating they have been accepted.

status
Displays current status of **ftp**.

struct [*structure*]
Sets data transfer structure type. The only structure supported is `stream`.

sunique
Toggles creating unique file names for remote destination files during **put** and **mput** operations. If unique remote file names is off (the default), **ftp** overwrites remote files. Otherwise, if a remote file has the same name as specified for a remote destination file, the remote FTP server modifies the name of the remote destination file. Note that the remote server must support the STOU instruction.

tenex
Sets the file transfer type to that needed for TENEX machines.

type [*typename*]
Sets the file transfer **type** to *typename*. If *typename* is not specified, the current type is printed. The default type is network ASCII; the type **binary** can be more efficient.

user *username* [*password*] [*account*]
Identifies the local user as *username* to the remote FTP server. If *password* or *account* is not specified and the remote server requires it, **ftp** prompts for it locally. If *account* is required, **ftp** sends it to the remote server after the remote login process completes. Note that, unless auto-login is disabled by specifying `-n` on the command line, this process is done automatically for the initial connection to the remote server.

verbose
Toggles verbose mode. When verbose mode is on (the default), **ftp** displays all responses from the remote FTP server. Additionally, **ftp** displays statistics on all file transfers when the transfers complete.

Examples

1. The user smith is logged in on host1. This example shows how smith can log in on the foreign host host2, check the current working directory on host2, list the contents of the working directory, and then transfer the file testfile to tmp.testfile:

```
$ ftp host2
Connected to host2.
220 host2 FTP Server (IBM RT) ready.
Name (host2:smith):
331 Passwd required for smith
Password:
230 User smith logged in

ftp> binary
200 Type set to I

ftp> pwd
257 "u/smith" is current directory

ftp> ls
200 PORT command successful.
150 Opening data connection for /bin/ls (192.9.200.1,1026) (0 bytes)
pri
testfile
226 Transfer complete.

ftp> get testfile tmp.testfile
200 PORT command successful.
150 Opening data connection for testfile (192.9.200.1,1029) (1201 bytes)
226 Transfer complete.
local:tmp.testfile remote:testfile

ftp> quit
221 Goodbye.
$ _
```

2. The user smith is logged in on host1. This example shows how smith can log in as the user smith on the foreign host host2:

```
$ ftp host2
connected to host2
220 host2 FTP Server (IBM RT) ready.
Name (host2:smith):
331 Passwd required for smith
Password:
230 User smith logged in
ftp>
```

3. The user fred makes a typing error and tries to log in as the user miths:

```
$ ftp test
220 test FTP Server (IBM RT) ready.
Name (test:fred): miths
530 User miths unknown
ftp> user smith
331 Passwd required for smith
Password:
230 User smith logged in
ftp>
```

4. The user fred issues the **ftp** command without specifying a host name:

```
$ ftp
ftp> open host1
connected to host1
220 host1 FTP Server (IBM RT) ready.
Name (host1:fred):
331 Passwd required for fred
Password:
230 User fred logged in
ftp>
```

File

.netrc Auto-login file in the current or home directory.

Related Information

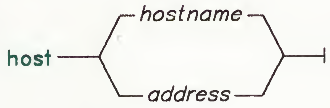
In this book: "**ftpd**" on page 3-5.

host

Purpose

Resolves a host name into an Internet address or an Internet address into a host name.

Syntax



A5ACC003

Description

The **host** command determines the Internet address when *hostname* is specified and the name of the host when *address* is specified. The *hostname* parameter can be either a unique host name or a well-known host name like **nameserver**, **printserver**, or **timeserver**. The *address* parameter must be a valid Internet address in dotted decimal format.

If the local host is using the DOMAIN protocol, the local or remote nameserver database is queried before searching the local **/etc/hosts** file.

Examples

1. To display the address of `host2`:

```
$ host host2
host2 is 192.100.13.5,  Aliases:  engr, samuel
$ _
```

2. To display the host whose address is `192.100.13.1`:

```
$ host 192.100.13.1
host7 is 192.100.13.1
$ _
```

File

/etc/hosts Defines the name-to-address map table.

Related Information

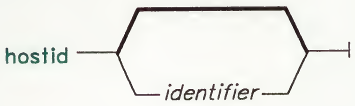
In this book: "**named**" on page 3-20.

hostid

Purpose

Sets or displays the identifier of the local host.

Syntax



A5ACC047

Description

The **hostid** command displays the *identifier* of the local host in hexadecimal or, if the command is issued by a user with superuser authority and an *identifier* argument is supplied, sets the identifier.

The ***identifier*** of a host can be either a unique host name or a numeric argument. This identifier must be unique across all hosts and is often set to the Internet address of the host. If the numeric argument is an Internet address, it can be entered in dotted decimal notation, and **hostid** converts it to hexadecimal. If desired, this command can be placed in the **/etc/rc.tcpip** initialization file.

Examples

1. To set the identifier of the local host, `host2`, to the local Internet address (if the user issuing the command has superuser authority):

   ```
   $ hostid 192.9.200.3
   0xc009c803
   $ _
   ```

2. To display the identifier of the local host, `host2`:

   ```
   $ hostid
   0xc009c803
   $ _
   ```

Related Information

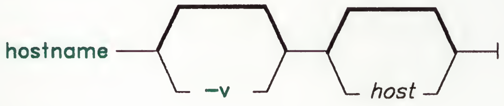
The **gethostid** and **sethostid** subroutines in *AIX Operating System Technical Reference.*

hostname

Purpose

Sets or displays the name of the local host system.

Syntax



A5ACC004

Description

Depending on the arguments you specify, the **hostname** command:

- displays the primary Internet name of the local host,
- displays the primary Internet name and address of the local host,
- sets and displays the primary Internet name of the local host if you have superuser authority.

If the *host* parameter is not specified, **hostname** displays the name of the local host. If the **-v** option is specified, **hostname** displays both the name and address of the local host. If you have superuser authority, you can set the primary name and address by supplying the *host* parameter. This is usually done in the startup script, **/etc/rc.tcpip**. The *host* parameter must be a name of no more than 32 characters and must be listed in the **/etc/hosts** file with an associated Internet address that is mapped to a network interface listed in your **/etc/net** file. **hostname** does not modify the **/etc/hosts** or **/etc/net** file.

If you have only one interface to the Internet network, the Internet address associated with that interface must be your primary Internet address. However, if your host is a ***gateway*** host (that is, a host that connects to more than one physical network), your host will have more than one Internet address. In this case, each network interface is defined in the **/etc/net** file with a different Internet address.

When using the DOMAIN protocol, a gateway host should have only one host name associated with all of its addresses and should not have any secondary names. Additionally, in a DOMAIN environment, the host name specified by the **hostname** command must be the full domain name for that host. Some programs, **sendmail** in particular, require that the name include the entire domain name.

Examples

1. To display the name of the local host:

```
$ hostname
host1
$ _
```

2. To display the name and address of the local host:

```
$ hostname -v
Node name: host1 Internet Address: 192.100.13.2
$ _
```

3. To change the name of the local host:

```
# hostname west
west
# _
```

File

/etc/hosts Defines the local name-to-address map table.

Related Information

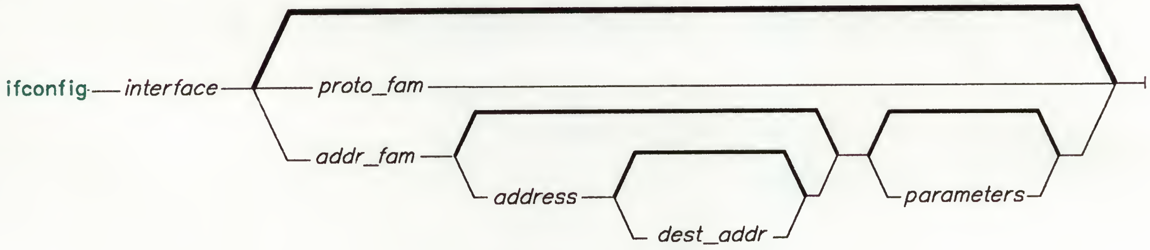
In this book: "**host**" on page 2-22.

ifconfig

Purpose

Configures or displays network interface parameters.

Syntax



A5ACC046

Description

The **ifconfig** command assigns an address to a network interface or configures network interface parameters. The **netconfig** command must be used at system startup to define the network address of each interface present on the system and to do an attach and open on the adapter. The **ifconfig** command can be used at other times to redefine an interface's address or to set other operating parameters.

An *interface* must be specified with the **ifconfig** command and is of the form:

```
name#
```

For example, the names `net0` or `token1` can be used. If no other parameters are supplied, the **ifconfig** command displays the current configuration for the specified network interface.

While any user can query the status of a network interface, only a user who has superuser privileges can modify the configuration of those interfaces.

If a *proto_fam* is specified, **ifconfig** reports only the configuration details specific to that protocol family.

When changing an interface configuration, an *address_family*, which may alter the interpretation of succeeding parameters, must be specified. This family is required because

an interface can receive transmissions in different protocols, each of which may require a separate naming scheme. The only supported address family is **inet**, which indicates the DARPA-Internet family. For the **inet** family, the *address* parameter is either a host name in the **/etc/hosts** database file, or an Internet address in the standard dotted decimal notation. For more information on the hosts database, see "**hosts**" on page 4-24.

The *dest_addr* argument specifies the address of the correspondent on the remote end of a point-to-point link.

Parameters

metric *n*
: Sets the routing metric, or number of hops, for the interface to the value of *n*. The default value is 0 if *n* is not specified, indicating that both hosts are on the same network. The routing metric is used by the **routed** daemon, with higher metrics indicating that the route is less favorable.

netmask *mask*
: Specifies how much of the address to reserve for subdividing networks into sub-networks. This parameter can only be used with an address family of **inet**. The *mask* includes the network part of the local address and the subnet part, which is taken from the host field of the address. The mask can be specified as a single hexadecimal number beginning with `0x`, in the standard Internet dotted-decimal notation, or with a name or alias that is listed in the **/etc/networks** file. The mask contains 1's for the bit positions in the 32-bit address that are reserved for the network and subnet parts, and 0's for the bit positions that specify the host. The mask should contain at least the standard network portion, and the subnet segment should be contiguous with the network segment.

trailers
: Requests the use of a *trailer* link-level encapsulation when sending messages.

-trailers
: Disables the use of a trailer link-level encapsulation. The use of trailers is disabled by default.

up
: Marks an interface as *up*. This parameter is used automatically when setting the first address for an interface, or can be used to enable an interface after an **ifconfig down** command.

down
: Marks an interface as *down*, which keeps the system from trying to transmit messages through that interface. If possible, the interface is reset to disable reception of messages as well. Routes that use the interface, however, are not automatically disabled.

arp Enables the use of the Address Resolution Protocol in mapping between network-level addresses and link-level addresses, and is currently used for mapping between DARPA Internet addresses and 10Mb/s Ethernet addresses. This parameter is on by default.

-arp Disables the use of the Address Resolution Protocol.

broadcast Specifies the address to use to represent broadcasts to the network. The default broadcast address is the address with a host part consisting of all 1's.

dstaddr Specifies the correspondent on the other end of a point-to-point link.

Examples

1. To query the status of serial line interface `tty0`, enter:

```
$ ifconfig tty0
tty0: flags=51<UP,POINTOPOINT,RUNNING>
        inet 192.9.201.3 --> 192.9.354.7 netmask ffffff00
$ _
```

2. To configure the local loopback interface (which is normally done automatically by the **/etc/rc.tcpip** file):

```
# ifconfig lo0 inet 127.0.0.1 up
# _
```

3. To mark the local Token-Ring interface, which is called `token0`, as down, enter the following:

```
# ifconfig token0 inet down
# _
```

   Only a user with superuser authority can modify the configuration of a network interface.

Related Information

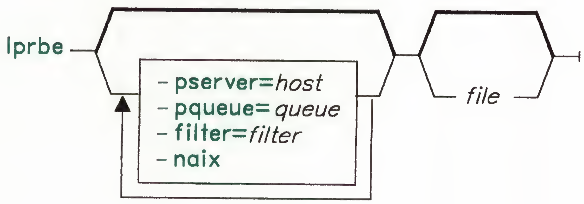
In this book: "**netstat**" on page 2-39, "**hosts**" on page 4-24, and "**networks**" on page 4-58.

lprbe

Purpose

Sends a file to a print server.

Syntax



A5ACC020

Description

The **lprbe** command is a backend program that sends print requests to a remote print server (the **lpd** program and printer on a foreign host). **lprbe** is normally called by the **qdaemon** command after you have queued a file with the **print** command. The flags and parameters that you enter with **print** are passed to **lprbe**, and it is **lprbe** that determines where and how the remote print job is done. **lprbe** supports the AIX **piobe** and **print** flags, as well as a set of filters that may exist on non-AIX systems. For more information on **piobe** and **print**, see *AIX Operating System Commands Reference.*

If a client requests a filter that is not listed in the print server's **/etc/filters** file, the print daemon, **lpd**, sends the file directly to the printer without processing (just as if the file were sent by the **cat** command).

Flags

-pserver = *host* — Specifies which host is to receive the print request. If **-pserver** = is not specified, **lprbe** sends the print request to the host specified by a **printserver** entry in the **/etc/hosts** file or to the remote print server identified in the **/etc/qconfig** file.

-pqueue = *queue* — Specifies a particular remote printing queue that is to receive the print request.

-filter=*filter* Specifies a *filter* that is either a user-defined program (with or without its own flags) that pipes its output to **print**, or one of the following flags (which indicate that the files to be printed are not standard text files). If *filter* contains embedded blanks, it must be enclosed in double quotes (" "). These flags generate a control file that is compatible with non-AIX systems; if they are sent to an AIX system, they are ignored.

- **-c** Handles files that contain data produced by **cifplot**.
- **-d** Handles files that contain T_EX data.
- **-f** Uses a filter that interprets the first character of each line as a standard FORTRAN carriage control character.
- **-g** Handles files that contain standard plot data as produced by the **plot** routines.
- **-l** Uses a filter that allows control characters to be printed and suppresses page breaks.
- **-n** Handles files that contain **ditroff** data.
- **-p** Uses **pr** to format the files. The results are equivalent to those obtained with the **print** command.
- **-r** Handles files that contain FORTRAN carriage control characters.
- **-t** Handles files that contain **troff** data.
- **-v** Handles files that contain a raster image for devices like the Benson Varian.

-naix Generates a print control file for a non-AIX system.

file Names one or more files to be printed.

Examples

1. To print the file `testcase` on a foreign AIX host (when `rp0` is the name of the local host queue that handles outbound print requests):

```
$ print rp0 testcase
$ _
```

2. To print on a foreign host (`host1`) other than the default **printserver**:

```
$ print rp0 -pserver=host1 testcase
$ _
```

3. To select a specific print queue (`lp1`) on a foreign host:

```
$ print rp0 -pqueue=lp1 testcase
$ _
```

4. To perform preprocessing on the foreign host (by the **pr** command) before printing:

```
$ print rp0 -filter="pr -w60 -i5" testcase
$ _
```

   This filter generates the following command on the foreign host:

```
pr -w60 -i5 testcase | print
```

5. To print on a non-AIX foreign host (that is configured as the default **printserver**):

```
$ print rp0 -naix testcase
$ _
```

File

/etc/qconfig Defines the queueing system configuration.

mail

Purpose

Sends and receives local or remote mail.

Related Information

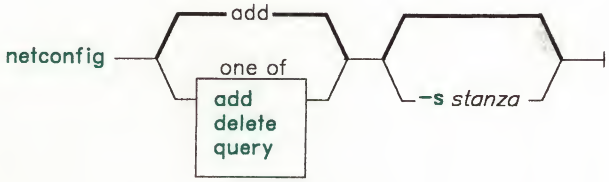
The **mail** command and the Message Handling (MH) commands in *AIX Operating System Commands Reference.*

netconfig

Purpose

Configures the specified interface or interfaces.

Syntax



A5ACC021

Description

The **netconfig** command processes the **/etc/net** file, establishing which adapter cards the Interface Program can communicate with. When issued without parameters, **netconfig** processes the entire **/etc/net** file and adds each stanza.

The **netconfig** command reads the specified stanza (or all stanzas) in the **/etc/net** file and, from the information provided in the stanza, configures the corresponding interface. The command returns a 0 exit status on success and a 1 if an error is encountered. Anything already configured at the time of the error remains configured. The **netconfig query** command can be issued by any user, but you must have superuser authority to add or delete an interface.

Flags

add Adds the interface specified by **-s** *stanza*. If **-s** *stanza* is not specified, **netconfig** processes all stanzas in the **/etc/net** file and adds those interfaces.

delete Deletes the interface specified by **-s** *stanza*. If **-s** *stanza* is not specified, **netconfig** processes all stanzas in the **/etc/net** file and deletes those interfaces.

query Returns the state of the interface specified in **-s** *stanza*. If **-s** *stanza* is not specified, **netconfig** returns the state of all interfaces defined by stanzas in **/etc/net**.

-s *stanza* Indicates that only the specified stanza, *stanza*, is to be processed.

Examples

1. To add all interfaces defined in **/etc/net**:

   ```
   # netconfig
   # _
   ```

   or

   ```
   # netconfig add
   # _
   ```

2. To add a specified interface, `net1`, that is defined in **/etc/net**:

   ```
   # netconfig add -s net1
   # _
   ```

3. To delete all interfaces defined in **/etc/net**:

   ```
   # netconfig delete
   # _
   ```

4. To delete a specified interface, `net1`, that is defined in **/etc/net**:

   ```
   # netconfig delete -s net1
   # _
   ```

5. To query all interfaces defined in **/etc/net**:

   ```
   # netconfig query

       net0:
       Internet Address:  192.100.13.1  inetlen:  1500
       hardware type=ethernet
       subnet mask: ffffff00    remote inetlen:  1500

       x25a0:
       Internet Address:  192.100.201.1  inetlen:  1007
       hardware type=x25
       subnet mask: ffffff00     remote inetlen:  576
   ```

```
    token0:
    Internet Address:  192.100.202.1  inetlen:  1568
    hardware type=IEEE 802.5
    subnet mask: ffff0000      remote inetlen:  1568
    broadcast:  local

    net1:
    Internet Address:  192.100.203.1  inetlen:  1500
    hardware type=802.3
    subnet mask: ffffff00      remote inetlen:  1500

    tty0:
    Internet Address:  192.100.204.1  inetlen:  576
    hardware type=Serial line
    subnet mask: ffffff00      remote inetlen:  576

    ip_security:
    level=Secret
    authority=GENSER,DSCCS-SPINTCOM
# _
```

6. To query a specified interface, net0:

```
# netconfig query -s net0
    net0:
    Internet Address:  192.100.13.1  inetlen:  1500
    hardware type=ethernet
    subnet mask: ffffff00     remote inetlen:  1500
```

Files

/etc/net Defines adapter cards for the Interface Program.

/etc/system Contains system device configurations.

Related Information

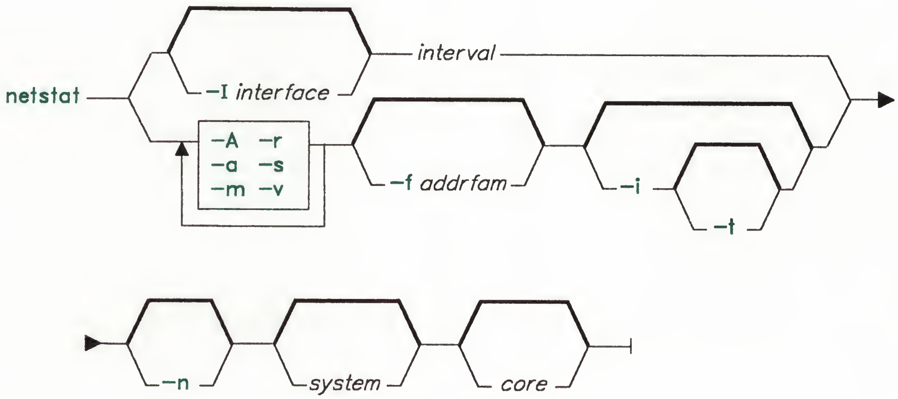
In this book: "**ifconfig**" on page 2-28 and "**net**" on page 4-52.

netstat

Purpose

Shows network status and provides problem determination information.

Syntax



A5ACC007

Description

The **netstat** command symbolically displays the contents of various network-related data structures for active connections. **netstat** accepts an *interval* argument, specified in seconds, that determines how frequently the requested interface status information should be displayed.

The *system* and *core* arguments allow alternatives to be specified for the default **/unix** and **/dev/kmem**, respectively.

The default display format shows the local and remote addresses, send and receive queue sizes in bytes, protocol, and the internal state of the protocol only for active sockets. Using the **-a** flag shows status for all sockets, not just active ones, while the **-A** shows protocol control block information for active sockets. Address formats are of the form *host.port* or *network.port* if the address of a socket specifies a network but not a particular host. When known, the host and network addresses are displayed symbolically according to the databases in **/etc/hosts** and **/etc/networks**, respectively. If a symbolic name for a

host is not known or if the **-n** flag is used, the address is printed numerically, according to the address family. Unspecified addresses and ports appear as an *.

The interface display format provides a table of cumulative statistics regarding packets transferred, errors, and collisions. The network addresses of the interface and the maximum transmission unit (`mtu`) are also displayed. When an *interval* is specified, **netstat** displays a running count of statistics related to network interfaces. This display consists of a column for the primary interface, which is the first interface found during configuration or the one specified with the **-I** flag, and a column summarizing information for all interfaces. The first line of each screen of information contains a summary since the system was last restarted, and subsequent lines of output show values accumulated over intervals of the specified length.

The routing table display format, using the **-r** or **-s** flags, indicates the available routes and their status. Each route consists of a destination host or network and a gateway to use in forwarding packets. The `flags` field shows the state of the route (`U` for up), whether the route is to a gateway (`G`), and whether the route was created dynamically by a redirect (`D`). Direct routes are created for each interface attached to the local host. The `gateway` field for these entries shows the address of the outgoing interface. The `refcnt` field gives the current number of active uses of the route. Connection-oriented protocols hold on to a single route for the duration of a connection, while connectionless protocols obtain a route while sending to the same destination. The `use` field provides a count of the number of packets sent using that route. The `interface` entry indicates the network interface utilized for the route.

Flags

-a Shows the state of all connections. Connections used by server processes are not shown without this flag.

-A Shows the address of any protocol control blocks associated with sockets. This flag is used for debugging purposes.

-f *address family* Limits reports of statistics or address control blocks to those of the specified *address family*. The following address families are recognized:

- **inet** Indicates the **AF_INET** address family.
- **unix** Indicates the **AF_UNIX** address family.

-i Shows the state of automatically configured connections. (Connections that are statically configured into the system, but not located at system start, are not shown.)

-I *interface* Shows information only about the *interface* specified. If this flag is used, an *interval* argument must also be entered.

-m Shows statistics recorded by the memory management routines.

-n Shows network addresses as numbers. When this flag is not specified, the **netstat** command interprets addresses, where possible, and displays them symbolically. The option can be used with any of the display formats.

-r Shows the routing tables. When used with the **-s** flag, the **-r** flag shows routing statistics along with the output of **-s**, instead of the routing tables.

-s Shows statistics for each protocol.

-t Adds timer information for automatically configured connections. This flag can only be used when the **-i** flag is also specified.

-v Shows VRM statistics for the IBM Baseband Adapter Device Driver, the IBM Token-Ring Adapter Device Driver, the X.25 Adapter Device Driver, and the 802.3 Adapter Device Driver.

Examples

1. To show the state of the connections that were automatically configured:

```
$ netstat -i
Name    Mtu   Network     Address    Ipkts  Ierrs Opkts Oerrs Collis
lo0     1536  127         127.0.0.1  12      0     0     0     0
token0  1536  128.114     host1      36606   0     0     0     0
net0    1500  192.9.200   host1      55      0     0     0     0
tty0    2048  192.9.210   slipa      0       0     0     0     0
$ _
```

2. To show the state of all active connections:

```
$ netstat -aA
PCB      Proto Recv-Q Send-Q  Local Address    Foreign Address  (state)
  124a8c tcp        0      0  *.time           *.*              LISTEN
  124c0c tcp        0      0  *.daytime        *.*              LISTEN
  124d8c tcp        0      0  *.chargen        *.*              LISTEN
  124f0c tcp        0      0  *.discard        *.*              LISTEN
  12508c tcp        0      0  *.echo           *.*              LISTEN
  125d8c tcp        0      0  *.finger         *.*              LISTEN
  12560c tcp        0      0  *.exec           *.*              LISTEN
  12570c tcp        0      0  *.login          *.*              LISTEN
```

```
  12588c tcp        0       0  *.shell         *.*              LISTEN
  125a0c tcp        0       0  *.telnet        *.*              LISTEN
  125b8c tcp        0       0  *.ftp           *.*              LISTEN
  12338c udp        0       0  *.*             *.*
  12458c udp        0       0  *.time          *.*
  12468c udp        0       0  *.daytime       *.*
  12478c udp        0       0  *.chargen       *.*
  12488c udp        0       0  *.discard       *.*
  12498c udp        0       0  *.echo          *.*
  12520c udp        0       0  *.ntalk         *.*
  125e8c udp        0       0  *.tftp          *.*
$ _
```

3. To show the routing tables:

```
$ netstat -r
Routing tables
Destination    Gateway      Flags   Refcnt Use   Interface
127.0.0.1      127.0.0.1    UH      0      0     lo0
default        host1        UG      0      0     net0
192.9.233      host15       U       0      9     token0
$ _
```

4. To show IBM Baseband Adapter and IBM Token-Ring Adapter Device Driver statistics:

```
$ netstat -v
      ************ VRM STATISTICS ************

net0:
hardware address: 00:dd:00:fe:70:00
Receive interrupts: 21  Packets accepted: 37   Packets Rejected:  0
Packets transmitted: 0
Collisions:  0  16_Collisions:  0   Shorted:  0   Underflow:  0
Short packets: 0  Alignment errors: 0 CRC errors: 0 Overflow: 0

token0:
hardware address:  7f:ff:81:00:07:5b
Receive interrupts:  45  Packets accepted:  47  Packets Rejected:  0
Transmit Interrupts:  47       Transmit Completes:  47
```

```
Packets transmitted:  47
Ring queue full:  0   SLIH ring empty:  0   Receive overflow:  0
```

Related Information

"**trpt**" on page 2-89, "**hosts**" on page 4-24, "**networks**" on page 4-58, "**protocols**" on page 4-60, and "**services**" on page 4-66.

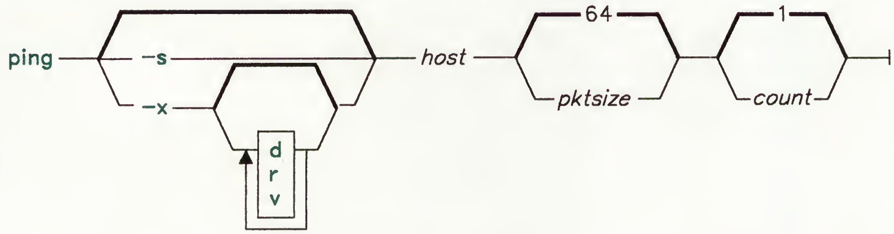
VRM Device Support and *VRM Programming Support.*

ping

Purpose

Sends an echo request to a network host.

Syntax



A5ACC008

Description

The **ping** command sends an ICMP **ECHO_REQUEST** to obtain an ICMP **ECHO_RESPONSE** from a host or gateway, and is useful for determining the status of the network and various foreign hosts. **ping** can also be used to track hardware and software problems, in network testing, measurement, and management, and for problem isolation.

The *host* parameter is either a valid host name or Internet address. If the host is operational and on the network, it responds to the echo. Each echo request contains an IP and ICMP header, followed by a **timeval** structure, and enough bytes to fill out the packet. The *pktsize* parameter indicates the number of bytes in each datagram, with 64 bytes used as the default. The optional *count* parameter specifies a number of echo requests to send, with the default value for *count* set to 1.

When trying to isolate a problem, first run **ping** on the local host to verify that the local interface to the network is up and running. Then use **ping** for hosts and gateways that are progressively more hops from the local host. The **ping** command sends one datagram per second and prints one line of output for every response received. **ping** calculates round-trip times and packet loss statistics, and displays a brief summary on completion. **ping** completes when *count* responses have been sent or the program times out, or on receipt of a **SIGINT** signal when no *count* is specified.

Because of the load continuous echo requests can place on the system, repeated requests should be used primarily for problem isolation.

Flags

-d Starts socket-level debugging.

-r Bypasses the routing tables and sends directly to a host on an attached network. If the *host* is not on a directly connected network, **ping** generates an error message. This option can be used to **ping** a local host through an interface that no longer has a route through it.

-s Requests output in a shortened form that includes only the name and address of the host and the phrase `Host responding`.

-v Requests verbose output, which lists ICMP packets that are received in addition to echo responses.

-x Indicates that **ping** should set the value of *count* to infinity and continue to send echo requests until a **SIGINT** signal is received.

Examples

1. To find out whether host `host1` is currently on the network:

```
$ ping -s host1
Ping host1:  192.100.13.2 responding
$ _
```

2. To obtain more complete statistics, such as round-trip time, about host `develop`:

```
$ ping develop
Ping develop:  56 data bytes
64 bytes from 192.9.201.3: icmp_seq=0, time=16. ms

----develop PING Statistics ----
1 packets transmitted, 1 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 16/16/16
$ _
```

Related Information

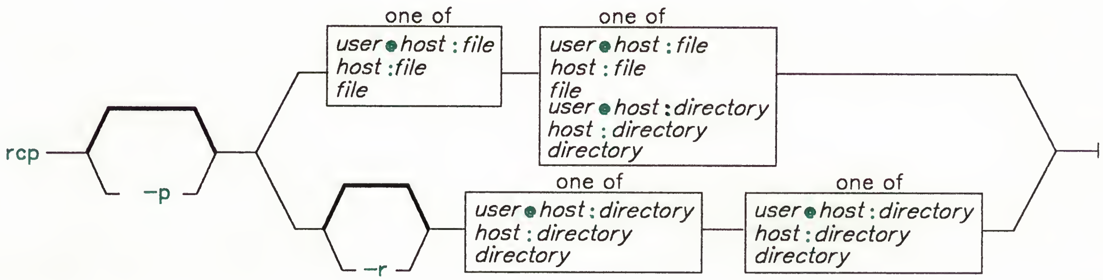
In this book: "**ifconfig**" on page 2-28, "**netconfig**" on page 2-35, and "**netstat**" on page 2-39.

rcp

Purpose

Copies files between a local and a foreign host or between two foreign hosts.

Syntax



A5ACC027

Description

The remote copy command **rcp** is used to copy one or more files between the local host and a foreign host, between two foreign hosts, or between files at the same foreign host. By default, the mode and owner of an existing destination file are preserved. Normally, if a destination file does not exist, the mode of the destination file is equal to the mode of the source file as modified by the **umask** at the destination host. If the **-p** flag is set, the modification time and mode of source files are preserved at the destination host. If a foreign host name is not specified for either the source or the destination, **rcp** is equivalent to the **cp** command.

When copying files to or from a foreign host, any remote file or directory name must be prefixed by the *host* of the foreign host and a colon (:). Local file and directory names do not need to have a *host* specified. However, since **rcp** assumes that a colon (:) terminates a host name, local file or directory names must have a slash (/) inserted before any colons embedded in the name.

The user name entered for the foreign host determines the file access privileges **rcp** uses at that host. Additionally, the user name given to a destination host determine the ownership and access modes of the resulting destination file or files. If a *host* is not prefixed by *user*@, the local user name is used at the foreign host. If a *user* is entered, that name is used. In either case, the foreign host allows access if one of the following conditions is satisfied:

- The local host is included in the foreign host's **/etc/hosts.equiv** file and the remote user is not the superuser.
- The local host and user name is included in a **.rhosts** file in the home directory of the remote user account.

For security reasons, any **.rhosts** file must be owned by either the remote user or root and only the owner can have read and write access.

In addition to the preceding conditions, **rcp** also allows access to the foreign host if the remote user account does not have a password defined. However, for security reasons, use of a password on all user accounts is recommended.

If the path for a file or directory on a foreign host is not specified or is not fully qualified, the path is interpreted as beginning at the home directory for the remote user account. Additionally, any metacharacters that must be interpreted at a foreign host must be quoted using \, ", or '.

Flags

-p Preserves the modification times and modes of the source files in the copies sent to the destination. Without this flag, the **umask** at the destination modifies the mode of the destination file, and the modification time of the destination file is set to the time the file is received.

-r Recursively copies, for directories only, each file and subdirectory in the source directory into the destination directory.

Examples

1. To copy a file named `localfile` from the local host to a foreign host named `host2`:

   ```
   $ rcp localfile host2:/u/eng/fred
   ```

2. To copy a remote file named `newplan` from one foreign host, `host1`, to another foreign host, `host2`:

   ```
   $ rcp host1:/u/eng/fred/newplan host2:/u/eng/mary
   ```

3. To send a directory subtree `report` from the local host to the home directory of a user named `fred` at a foreign host named `host2`, and preserve all modes and modification times:

   ```
   $ rcp -p -r report fred@host2:report
   ```

   The remote file `/u/fred/.rhosts` includes an entry specifying the local host and user name.

Files

/etc/hosts.equiv	Defines equivalent hosts.
$HOME/.rhosts	Defines equivalent remote users.

Related Information

In this book: "**rsh, remsh**" on page 2-60, "**rlogin**" on page 2-54, and "**rshd**" on page 3-33.

remsh

Purpose

Logs into a foreign host or executes the specified command at the foreign host.

Related Information

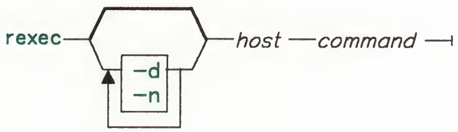
For a full description of this command, see "**rsh**, **remsh**" on page 2-60.

rexec

Purpose

Executes commands one at a time on a foreign host.

Syntax



A5ACC022

Description

The **rexec** command executes a command on the specified foreign host. The *host* parameter specifies the name of the host where the command is to be executed in alphanumeric form. The *command* parameter specifies the command, including any flags or parameters, to be executed on the foreign host. If *command* contains embedded blanks, it should be enclosed in double quotes (" "). The **rexec** command sends and receives data over a TCP connection.

rexec provides an automatic login feature by checking for a **$HOME/.netrc** file that contains the user name and password to use at the foreign host. If such an entry is not found or if your system is operating in controlled access mode, **rexec** prompts for a valid user name and password for the foreign host. You can also override the automatic login feature by specifying the **-n** flag on the **rexec** command line.

Flags

-d Enables socket-level debugging.

-n Prevents automatic login. With the **-n** flag, **rexec** prompts for a user name and password to use at the foreign host, rather than searching for a **$HOME/.netrc** file.

Note: In versions of AIX prior to 2.2.1, the **-n** flag to **rexec** searched for a **$HOME/.netrc**. In Version 2.2.1 and subsequent releases, **rexec** searches for the **.netrc** file by default, and the **-n** flag *prevents* the search.

Examples

1. To execute the **date** command on foreign host **host1**, using a **$HOME/.netrc** file on the local host that contains a user name and password valid at the foreign host:

   ```
   $ rexec host1 date
   [date command output]
   $ _
   ```

2. To list the directory of user **tom** on foreign host **host1**:

   ```
   $ rexec host1 "li -l /u/tom"
   Name(host1:fred):  tom
   Passwd(host1:tom):  <enter password>
   [listing of tom's directory on foreign host]
   $ _
   ```

3. To list the **/tmp** directory of foreign host `host1`:

   ```
   $ rexec host1 "li /tmp"
   Name(host1:fred):  tom
   Passwd(host1:tom):  <enter password>
   [listing of /tmp on foreign host]
   ```

4. To override the automatic login feature and execute a command as user name `tom` on foreign host `host1`, enter:

   ```
   $ rexec -n host1 date
   Name(host1:fred):  tom
   Password: <Enter password>
   [date command output]
   $ _
   ```

File

$HOME/.netrc Contains user names and passwords for remote login.

Related Information

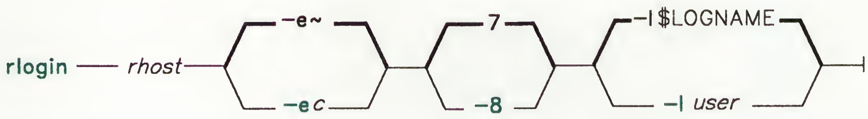
In this book: "**rexecd**" on page 3-24.

rlogin

Purpose

Connects the local host with a foreign host.

Syntax



A5ACC031

Description

The remote login command **rlogin** logs into the foreign host *rhost* and connects your local terminal to the foreign host. The remote terminal type is the same as that given in the local environment variable **TERM**. The terminal or window size is also the same, if the foreign host supports them, and any changes in size are transferred. All echoing takes place at the foreign host, so except for delays, the terminal connection is transparent. **Ctrl-S** and **Ctrl-Q** stop and start the flow of information, and the input and output buffers are flushed on interrupts.

If you do not specify the **-l** flag, the local user name is used at the foreign host. If `-l` *username* is specified, the user name entered is used at the foreign host. In either case, the remote host allows access only if one or both of the following conditions is satisfied:

- The local host is included in the foreign host's **/etc/hosts.equiv** file, the local user is not the superuser, and the **-l** *username* flag is not specified.
- The local host and user name is included in a **.rhosts** file in the home directory of the remote user account.

If neither of these conditions is met and a password is defined for the remote user account, the foreign host prompts for a password. The remote password file is checked to verify the password entered, and the login prompt is displayed if the password is not correct. Entering **Ctrl-D** at the login prompt ends the remote login attempt.

For security reasons, any **.rhosts** file must be owned by either the remote user or root and should allow write access only by the owner.

In addition to the preceding conditions, **rlogin** also allows access to the foreign host if the remote user account does not have a password defined. However, for security reasons, use of a password on all user accounts is recommended.

Unless otherwise modified by the **-e** flag, the standard escape character is a tilde (~). The escape character is only recognized by the foreign host if it occurs at the beginning of a line. Otherwise, the escape character is sent to the foreign host as a normal character. To send the escape character to the foreign host as a normal character at the beginning of a line, press the escape character twice. Pressing the escape character and a period (for example, `~.`) immediately disconnects the local terminal from the foreign host.

Flags

-e*c* Changes the escape character. Substitute the character you choose for *c*.

-8 Allows an 8-bit data path at all times. Otherwise, unless the start and stop characters on the foreign host are not Ctrl-S and Ctrl-Q, **rlogin** uses a 7-bit data path and parity bits are stripped.

-l *user* Changes the remote user name to the one you specify. Otherwise, your local user name is used at the foreign host.

Examples

1. To connect your local terminal to the foreign host `host2`, logging in to a user account with the same name as your local user account:

   ```
   $ rlogin host2
   Password: <Enter password>
   ```

2. To log in to `host2` with the same name as your local user account and change the escape character to \:

   ```
   $ rlogin host2 -e\
   ```

   In this example, the local host is listed in the **/etc/hosts.equiv** file at the remote host.

3. To use the user name `fred` at the foreign host:

   ```
   $ rlogin host2 -l fred
   Password: <Enter password>
   ```

Files

/etc/hosts.equiv	Defines equivalent hosts.
$HOME/.rhosts	Defines equivalent remote users.

Related Information

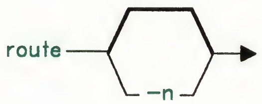
In this book: "**rcp**" on page 2-47, "**rsh, remsh**" on page 2-60, and "**rlogind**" on page 3-26.

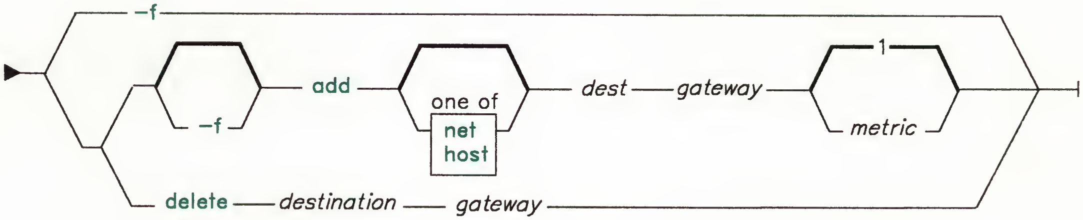
route

Purpose

Manually manipulates the routing tables.

Syntax





A5ACC009

Description

The **route** command manually manipulates the network routing tables. You must have superuser authority or be a member of the system group to run the **route** command. The command accepts two subcommands:

add Adds a route.

delete Deletes the specified route.

The interpretation of the network address associated with the *destination* distinguishes routes to hosts from those to networks. The optional **net** or **host** parameter can be used to indicate that the *destination* should be interpreted as a network or a host, respectively. If neither of these parameters is specified and the host portion of the address is 0 (that is, only the network portion is specified), the route is assumed to be to a network. Otherwise, the route is assumed to be to a host.

The *destination* parameter names a host or network where the route is directed, and the *gateway* parameter names the gateway to which the packets should be addressed. These parameters can be specified either by symbolic name or numeric address. All symbolic names are resolved into addresses using either the **/etc/hosts** file or the network name server.

The *metric* indicates the number of hops to the destination gateway and is required for **add** commands. The value of *metric* must be 0 for a destination on a directly attached network, and greater than 0 if the route uses one or more gateways. If adding a route with metric 0, the gateway given is the address of this host on the common network, indicating the interface to be used for transmission. For more information on metrics, see "**gateways**" on page 4-21.

The **netstat -r** command shows the information contained in the routing tables. For more information on how to use this command, see "**netstat**" on page 2-39.

Flags

-f Provides a flush option, which clears the routing table of all gateway entries before making the specified change, if any.

-n Displays host and network names numerically, rather than symbolically, when reporting actions.

Examples

1. To establish a route for sending a message to host address `192.100.201.7`, which is not on the same network as the host sending the message:

```
$ route add 192.100.201.7 192.100.13.7
$ _
```

2. To establish a route that will enable you to send a message to any user on network `192.100.201`:

```
$ route add 192.100.201.0 192.100.13.7
$ _
```

3. To establish a default gateway, 192.100.13.7:

```
$ route add 0 192.100.13.7
$ _
```

The value 0 for the destination means that any packets that are sent to destinations not on a directly connected network and not previously defined go through the default gateway.

4. To clear the host gateway table:

```
$ route -f
$ _
```

5. To clear the host gateway table, then add a default gateway:

```
$ route -f add 0 192.100.13.7
$ _
```

Related Information

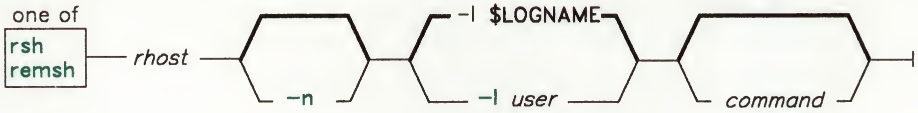
In this book: "**routed**" on page 3-28.

rsh, remsh

Purpose

Logs into a foreign host or executes the specified command at the foreign host.

Syntax



A5ACC030

Description

The remote shell command **rsh** executes *command* at *rhost* or, if no *command* is specified, logs into *rhost*. **rsh** sends standard input from the local command line to the remote command and receives standard output and standard error from the remote command. The **remsh** command performs the same function as the **rsh** command. In the rest of this section, the name **rsh** refers to either the **rsh** or **remsh** command.

Since any input to the remote command must be specified on the local command line, you cannot use **rsh** to execute an interactive command on a foreign host. If you need to execute an interactive command on a foreign host, use either **rlogin** or **rsh** with no *command* specified. If you do not specify a *command*, **rsh** executes **rlogin** instead.

If you do not specify the **-l** flag, the local user name is used at the foreign host. If -l *username* is entered, the specified user name is used at the foreign host. In either case, the remote host allows access only if at least one of the following conditions is satisfied:

- The local user ID is not superuser, and the name of the local host is listed as an equivalent host in the remote **/etc/hosts.equiv** file.
- If either the local user ID is superuser or the check of **/etc/hosts.equiv** fails, the remote user's home directory must contain a **.rhosts** file that lists the local host and user name.

For security reasons, any **.rhosts** file must be owned by either the remote user or root, and only the owner should have read and write access.

In addition to the preceding conditions, **rsh** also allows access to the foreign host if the remote user account does not have a password defined. However, for security reasons, use of a password on all user accounts is recommended.

While the remote command is executing, pressing the INTERRUPT, TERMINATE, or QUIT key sends the corresponding signal to the remote process. However, pressing the STOP key stops only the local process. Normally, when the remote command terminates, the local **rsh** process terminates.

To have shell metacharacters interpreted on the foreign host, place the metacharacters inside double quotes (" "). Otherwise, the metacharacters are interpreted by the local shell.

If the file *rhost* exists, is linked to **rsh**, and is in your command search path, you can omit `rsh` in the remote shell command. In this case, the command is:

rhost `[-n]` `[-l` *username*`]` *command*

where brackets identify optional items.

Flags

-n Redirects any input for **rsh** to the **/dev/null** device. Use this flag if you are in C shell and run **rsh** in the background.

-l *user* Specifies that **rsh** should log into the foreign host as the named *user* instead of the local user name. If this flag is not specified, the local and remote user names are the same.

Examples

In the following examples, the local host (`host1`) is listed in the **/etc/hosts.equiv** file at the remote host (`host2`).

1. To check the amount of free disk space on the foreign host `host2`:

   ```
   $ rsh host2 df
   ```

2. To append a remote file to another file on the foreign host, put the metacharacters in quotation marks:

   ```
   $ rsh host2 cat remotefile1 ">>" remotefile2
   ```

3. To append a remote file at the foreign host to a local file, omit the quotation marks:

   ```
   $ rsh host2 cat remotefile >> localfile
   ```

4. To append a remote file to a local file and use a remote user's permissions at the remote host, use the **-l** flag:

```
$ rsh host2 -l fred cat remotefile >> localfile
```

Files

/etc/hosts.equiv	Defines equivalent hosts.
$HOME/.rhosts	Defines equivalent remote users.

Related Information

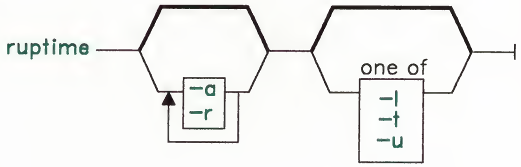
In this book: "**rcp**" on page 2-47, "**rlogin**" on page 2-54, and "**rshd**" on page 3-33.

ruptime

Purpose

Shows the status of each host on a network.

Syntax



A5ACC028

Description

The **ruptime** command displays the status of each host that is on a local network and is running the **rwhod** daemon. The status lines are sorted by host name unless the **-l**, **-t**, or **-u** flag is indicated. The status information is provided in packets broadcast once every 3 minutes by each network host running **rwhod**. Any activity (such as power to a host being turned on or off) that takes place between broadcasts is not reflected until the next broadcast. Hosts for which no status information is received for 11 minutes are reported as down.

Flags

-a Includes all users. Without this flag, users whose sessions are idle an hour or more are not included.

-l Sorts the list by the load average.

-r Reverses the sort order.

-t Sorts the list by the uptime.

-u Sorts the list by the number of users.

Examples

1. To get a status report on the hosts on the local network:

```
$ ruptime

host1          up      5:15,   4 users,        load 0.09, 0.04, 0.04
host2          up      7:45,   3 users,        load 0.08, 0.07, 0.04
host3          up      2:28,   0 users,        load 0.01, 0.02, 0.03
host4          up   3+01:44,   1 user,         load 0.01, 0.02, 0.03
host7          up      7:43,   1 user,         load 0.06, 0.12, 0.11
```

2. To get a status report sorted by load average:

```
$ ruptime -l

host2          up      7:45,   3 users,        load 0.08, 0.07, 0.04
host1          up      5:18,   4 users,        load 0.07, 0.07, 0.04
host7          up      7:43,   1 user,         load 0.06, 0.12, 0.11
host3          up      2:28,   0 users,        load 0.01, 0.02, 0.03
host4          up   3+01:44,   1 user,         load 0.01, 0.02, 0.03
```

File

/usr/spool/rwho/whod.* Indicates data files received from remote **rwhod** daemons.

Related Information

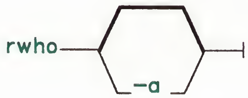
In this book: "**rwho**" on page 2-65 and "**rwhod**" on page 3-35.

rwho

Purpose

Shows which users are logged in to hosts on the local network.

Syntax



A5ACC029

Description

The **rwho** command displays the user name, host name, and start date and time of each session for everyone on the local network who is currently logged in to a host running the **rwhod** daemon. If a user has not typed anything for at least 3 minutes, **rwho** reports the idle time as a number of minutes in the last column. After an hour of inactivity, a user is not included unless the **-a** flag is specified. Use this command with caution if the local network has a large number of users.

Status information is broadcast once every 3 minutes by each network host running **rwhod**. Any activity (such as a user logging on or off) that takes place between broadcasts is not reflected until the next broadcast.

Flag

-a Includes all users. Without this flag, users whose sessions are idle an hour or more are not included in the report.

Example

To get a report of all users currently logged into hosts on the local network:

```
$ rwho

bob     host2:pts5       Nov 17 06:30 :20
bob     host7:console    Nov 17 06:25 :25
fran    host1:pts0       Nov 17 11:20 :51
fran    host1:pts8       Nov 16 15:33 :42
fran    host4:console    Nov 17 16:32
thomas  host1:console    Nov 17 13:14 :31
thomas  host1:pts7       Nov 17 13:15 :47
server  host2:console    Nov 17 06:58 :20
alice   host2:pts6       Nov 17 09:22
```

File

/usr/spool/rwho/whod.* Indicates data files received from remote **rwhod** daemons.

Related Information

In this book: "**ruptime**" on page 2-63 and "**rwhod**" on page 3-35.

securetcpip

Purpose

Enables controlled access mode, the AIX networking security feature.

Syntax

```
securetcpip ——|
```

A5ACC042

Description

The **securetcpip** command enables controlled access mode, which provides enhanced networking security. This command disables programs that are not trusted, which includes **rcp**, **rlogin**, **rsh**, **tftp**, **trpt**, and their associated daemons. In addition, **securetcpip** sets up the secure version of the interface program and adds a stanza to the **/etc/security** configuration file that disables the **.netrc** autologin feature for the **ftp** and **rexec** commands.

After issuing the **securetcpip** command, you should shut down your system, then restart it. All of your TCP/IP commands and network interfaces will be properly configured after the system restarts.

Related Information

In this book: "**.netrc**" on page 4-68.

The **secure** command in *AIX Operating System Commands Reference.*

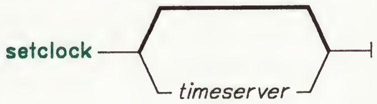
The discussion of *trusted programs* in *Managing the AIX Operating System.*

setclock

Purpose

Sets the time and date.

Syntax



A5ACC010

Description

The **setclock** command gets the time from a network time server, and if run by the superuser, sets the local RT time and date accordingly. **setclock** is designed for use either as a stand-alone command or as a command that can be invoked in the **/etc/rc.tcpip** command file.

The **setclock** command sends an Internet TIME service request to a time server host. The *timeserver* parameter can be either a host name or address of a network host that services TIME requests. If the *timeserver* name is omitted, **setclock** sends the request to the default time server, **timeserver**, specified by the name server in a DOMAIN environment or by the **/etc/hosts** file otherwise.

Note: Any host running the **inetd** daemon can act as a time server.

setclock takes the first response from the time server, converts the calendar clock reading found there, and displays the local date and time. Finally, if **setclock** is run by a user with superuser authority, it calls the standard RT entry points to set the system date and time.

If no time server responds, or the network is not operational, **setclock** displays a message to that effect and leaves the current date and time settings of the system unchanged.

Examples

1. To display the date and time using the `timeserver` host specified in **/etc/hosts**:

```
$ setclock
Sat Mar 11 15:31:05 1988
$ _
```

2. To set the date and time from the time server in `host1`:

```
$ su
# setclock host1
Sat Mar 11 15:32:27 1988

# _
```

Related Information

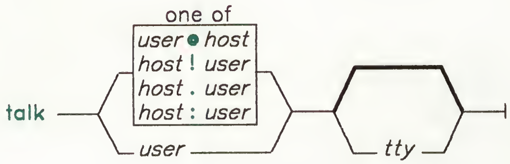
In this book: "**inetd**" on page 3-13.

talk

Purpose

Converse with another user.

Syntax



A5ACC032

Description

The **talk** command allows two users to type simultaneously into windows displayed on each other's terminals. To initiate a conversation, a user executes **talk** and specifies the second user's account name. If the second user is on a foreign host, the name of the host must also be specified in one of the following ways:

user@host
host!user
host.user
host:user

When using full domain names, the only valid form for specifying the user and host is *user@host*. For example, `norman@host17.dev.ibm.com` initiates a conversation with user `norman` at host `host17` in the `dev.ibm.com` domain.

When the first user initiates the conversation, a message is sent to the second user, inviting a conversation. If the first user also specifies *ttyname*, the invitation message is sent only to the specified terminal. Otherwise, the invitation is sent to the console on the foreign host where the second user is logged in. Once this invitation is received, **talk** displays two windows on the first user's terminal and displays progress messages until the second user responds to the invitation.

If the second user wants to have the conversation, the second user also executes **talk** from any terminal and specifies the first user's account name and host name, if appropriate. If the second user accepts the invitation, **talk** displays two windows on each user's terminal.

One window displays what is typed by the local user; the other, what is typed by the remote user. To end the conversation, either user can press the INTERRUPT key and the connection is closed.

If the second user does not want to permit **talk** invitations, that user should issue the **mesg n** command. For more information on using the **mesg** command, see *AIX Operating System Commands Reference.*

Note: The **talk** command uses the talk 4.3 protocol, which is not compatible with 4.2 versions of **talk**.

Examples

1. If john at host1 wants to talk to fred, who is logged in on host2, john enters:

   ```
   $ talk fred@host2
   ```

   The following message is displayed on fred's terminal:

   ```
   Message from TalkDaemon@host1 at 15:16...
   talk: connection requested by john@host1.
   talk: respond with: talk john@host1
   ```

   To accept the invitation, fred enters:

   ```
   $ talk john@host1
   ```

2. To talk to fred only if he is logged in on the console at host2, enter:

   ```
   $ talk fred@host2 console
   ```

Related Information

In this book: "**talkd**" on page 3-39.

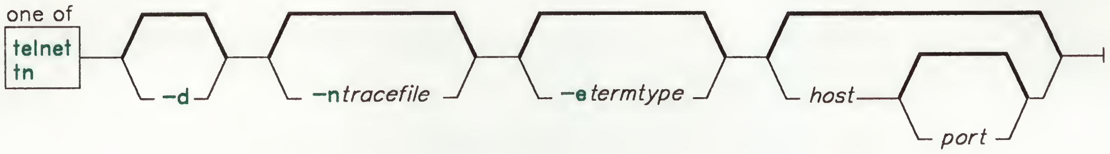
The **mesg** command in *AIX Operating System Commands Reference.*

telnet, tn

Purpose

Provides the TELNET interface for logging in to a foreign host.

Syntax



A5ACC014

Description

The **telnet** command implements the TELNET protocol, which allows remote login to other hosts. It uses the Transmission Control Protocol (TCP) to communicate with other hosts in the network. The **tn** command is identical to the **telnet** command, and in the following discussion, **telnet** means either the **telnet** command or the **tn** command.

The **telnet** command operates in two different modes: ***command mode*** and ***input mode***. When issued without arguments, **telnet** enters command mode, as indicated by the `telnet>` or `tn>` prompt. Command mode can also be entered by pressing **Ctrl-t** in input mode.

In command mode, the subcommands listed under "Subcommands" on page 2-75 can be entered. Some of these subcommands return you to the remote session upon completion. For those that do not, pressing **Enter** returns you to the remote session.

If **telnet** is issued with arguments, it performs an **open** subcommand with those arguments, then enters input mode. The type of input mode is either ***character-at-a-time*** or ***line-by-line***, depending on what the remote system supports.

In character-at-a-time mode, most text typed is immediately sent to the remote host for processing. In line-by-line mode, all text is echoed locally and completed lines are sent to the remote host. The local echo character is used to turn off and on the local echo, and its initial value is **E**.

In either input mode, if **toggle localchars** is **true** (see "Subcommands" on page 2-75), the user's **QUIT**, **INTR**, and **FLUSH** characters are trapped locally and sent as Telnet

Protocol sequences to the remote host. The **toggle autoflush** and **toggle autosynch** subcommands cause this action to flush subsequent output to the terminal until the remote host acknowledges the TELNET sequence, and to flush previous terminal input (in the case of **QUIT** and **INTR**).

To enter **telnet** command mode while connected to a remote host, type the TELNET escape key sequence. The default escape sequence is **Ctrl-t**. When in command mode, the standard AIX editing conventions, such as backspace, are available.

For the ptys that are to be used by TELNET for remote login to this host with the **telnet** command, use the **devices** command to set the value of **terminal type** as required and the values of **logger** and **automatic enable** to **true**. (**automatic enable** causes the **pstart** command to be run for that device at system start, making it available for use by TELNET.)

The **telnet** command supports an option called *terminal negotiation*. If the remote host supports terminal negotiation, **telnet** sends the local terminal type to the remote host. If the remote host does not accept the local terminal type, **telnet** attempts to emulate a 3270 terminal and a DEC VT100[2] terminal. If you have specified a terminal to emulate, **telnet** does not negotiate for terminal type. If the local and remote hosts cannot agree on a terminal type, the local host defaults to vt100.

To override the terminal negotiation from the console, use the **EMULATE** environment variable or the **-e** option. To determine whether terminal type negotiation is performed, the following list describes the order of the **telnet** command processing:

1. The **-e** command line flag. (No negotiation)
2. The **EMULATE** environment variable. (No negotiation)
3. If items 1 and 2 are not present, terminal type negotiation occurs automatically.

If the client and the server negotiate to use a 3270 data stream, the keyboard mapping to be used is determined by the following precedence:

$HOME/.3270keys	User's 3270 keyboard mapping
/etc/3270.keys.rt	Standard RT 3270 keyboard (console) mapping
/etc/3270.keys	Base 3270 keyboard mapping for use with limited function terminals.

telnet supports these 3270 terminal types: 3277-1, 3278-1, 3278-2, 3278-3, 3278-4, and 3278-5. If you are using **telnet** in 3270 mode on a color display, the colors and fields are displayed the same as those of an actual 3279 display by default. You can select other colors by editing one of the keyboard mapping files in the preceding list. When the **telnet** session is ended, the display is reset to the colors in use before the session began.

In addition to terminal negotiation, **telnet** allows negotiation for the Secure Attention Key (SAK) option. This option, when supported, provides the local user with a secure

---

[2] VT100 is a trademark of Digital Equipment Corporation.

communication path to the remote host for tasks such as changing user IDs or passwords. If the remote host supports the SAK function, a trusted shell is opened on the remote host when the **telnet send sak** command is issued. The SAK function can also be assigned to a single key available in **telnet** input mode, using the **set sak** command as described in the section on "Subcommands" on page 2-75.

International Character Support Considerations

The TELNET protocol treats the decimal value 255 (hex FF) as a flag indicating that a control value follows. Therefore, the following characters can produce unpredictable results if sent across a TELNET connection:

n acute (lowercase)

∴ (the ***therefore*** symbol).

Restrictions

The mouse cannot be used as an input device with **telnet**.

telnet does not support the APL data stream.

Environment Variables

The following environment variables can be used with the **telnet** command:

EMULATE Overrides terminal type negotiation in the same way as the **-e** flag. If the value of the **EMULATE** environment variable is defined as **vt100** or **3270**, **telnet** emulates a DEC VT100 terminal or 3270 terminal, respectively. If **EMULATE** is not defined or has a value of **none**, **telnet** operates normally. **EMULATE** is valid only for the console; if **EMULATE** is set for other than the RT console, results may be unpredictable. If **EMULATE** is set to **vt100** or **3270**, the **TERM** environment variable in the remote login connection should be set to the same value. (Check this with the **env** command after the connection is open.)

TNESC Specifies an alternate TELNET escape character, other than the default **Ctrl-t**. To change the **telnet** escape sequence, set **TNESC** to the octal value of the character you want to use. Then export **TNESC**. For example, set **TNESC** to 35 to change the TELNET escape sequence to **Ctrl-]**. See *AIX Operating System Technical Reference* for the table that maps octal values to their ASCII equivalents.

For more information on using environment variables, see the **env** command in *AIX Operating System Commands Reference.*

Flags

-d Turns debugging mode on.

-e *termtype* Overrides terminal type negotiation. Possible values are: **vt100**, **3270**, or **none**.

-n *tracefile* Records network trace information in the file specified by *tracefile*.

? Displays command syntax for available **telnet** commands.

Subcommands

For each of the subcommands in the following list, you only need to type enough letters to uniquely identify the command. (For example, **q** is sufficient for the **quit** command.) This is also **true** for the arguments to **emulate**, **display**, **mode**, **set**, and **toggle**.

The subcommands for **telnet** and **tn** are:

? [*command***]**
Requests help on **telnet**. Without arguments, **telnet** prints a help summary. If a *command* is specified, **telnet** prints help information for just that command.

close Closes the TELNET connection and returns to command mode.

display [*argument***]**
Displays all of the **set** and **toggle** values if no *argument* is specified; otherwise, lists only those values that match *argument*.

emulate *terminaltype*
Overrides terminal type negotiation with the specified *terminaltype*. Possible choices are:

? Prints help information.

3270 Emulates a 3270 terminal.

none Specifies no emulation.

vt100 Emulates a DEC VT100 terminal.

All output received from the remote host is processed by the specified emulator. The initial terminal type to emulate can be specified through the **EMULATE** environment variable or the **-e** option to the **telnet** command.

mode *type*
Specifies the current input mode. When *type* is **line**, the mode is line-by-line. When *type* is **character**, the mode is character-at-a-time. Permission is requested from the remote host before entering the requested mode, and if the remote host supports it, the new mode is entered.

open *host* [*port*]
Opens a connection to the specified *host*. The host specification can be either a host name or an Internet address in dotted decimal form. If no *port* is given, **telnet** attempts to contact a TELNET server at the default port.

quit Closes a TELNET connection and exits **telnet**. An **END OF FILE** in command mode also closes the connection and exits.

send *arguments*
Sends one or more arguments (special character sequences) to the remote host. Multiple arguments are separated by spaces. The following arguments can be used:

? Prints help information for the **send** command.

ao Sends the TELNET AO (Abort Output) sequence, which causes the remote host to flush all output from the remote system to the local terminal.

ayt Sends the TELNET AYT (Are You There) sequence, to which the remote system can respond.

brk Sends the TELNET BRK (Break) sequence, which causes the remote system to perform a kill operation.

ec Sends the TELNET EC (Erase Character) sequence, which causes the remote host to erase the last character entered.

el Sends the TELNET EL (Erase Line) sequence, which causes the remote system to erase the line currently being entered.

escape Sends the current **telnet** escape character (**Ctrl-t** by default).

ga Sends the TELNET GA (Go Ahead) sequence, which provides the remote system with a mechanism to signal the local system to return control to the user.

ip Sends the TELNET IP (Interrupt Process) sequence, which causes the remote system to cancel the currently running process.

nop Sends the TELNET NOP (No Operation) sequence.

sak Sends the TELNET SAK (Secure Attention Key) sequence, which causes the remote system to invoke the trusted shell. If the SAK is not supported, then an error message appears that reads: `Remote side does not support SAK.`

synch Sends the TELNET SYNC sequence, which causes the remote system to discard all previously typed input that has not yet been read. This sequence is sent as TCP urgent data.

set *variable value*

Sets a TELNET *variable* to the specified *value*. The special value **off** turns off the function associated with the variable named entered. The **display** command can be used to query the current setting of each variable. The variables that can be specified are:

echo
: Toggles between local echo of entered characters and suppressing local echo. Local echo is used for normal processing, while suppressing the echo is used for entering text that should not appear on the display, such as passwords. This variable can only be used in line-by-line mode.

eof
: Defines the **END OF FILE** character for **telnet**. When **telnet** is in line-by-line mode, entering the **eof** character as the first character on a line sends the character to the remote host. The initial value for the **eof** character is the local terminal's **END OF FILE** character.

erase
: Defines the erase character for **telnet**. When **telnet** is in character-at-a-time mode and **localchars** is **true**, typing the **erase** character sends the TELNET EC sequence to the remote host. The initial value for the **erase** character is the local terminal's **ERASE** character.

escape
: Specifies the **telnet escape** character, which puts **telnet** into command mode when connected to a remote host. This character can also be specified in octal in the **TNESC** environment variable.

flushoutput
: Defines the flush character for **telnet**. When **localchars** is **true**, typing the **flushoutput** character sends the TELNET AO sequence to the remote host. The initial value for the flush character is **Ctrl-O**. If the remote host is running AIX, **flushoutput**, unlike the other special characters defined by **set**, only works in **localchars** mode since it has no **termio** equivalent.

interrupt
: Defines the interrupt character for **telnet**. When **localchars** is **true**, typing the **interrupt** character sends the TELNET IP sequence to the remote host. The initial value for the **interrupt** character is the local terminal's **INTERRUPT** character.

kill
: Defines the kill character for **telnet**. When **telnet** is in character-at-a-time mode and **localchars** is **true**, typing the **kill** character sends the TELNET EL sequence to the remote host. The initial value for the **kill** character is the local terminal's **KILL** character.

quit Defines the quit character for **telnet**. When **localchars** is **true**, typing the **quit** character sends the TELNET BRK sequence to the remote host. The initial value for the **quit** character is the local terminal's **QUIT** character.

sak Defines the Secure Attention Key (SAK) for **telnet**. When the **sak** character is entered, the remote system is asked to create a trusted shell. If the remote host does not support the SAK, this sequence has no effect.

status Shows the status of **telnet**, including the current mode and the currently connected remote host.

toggle *arguments*
Toggles one or more arguments that control how **telnet** responds to events. Possible values are **true** and **false**. Multiple arguments are separated by spaces. The **display** command can be used to query the current setting of each argument. The following arguments can be used:

? Displays valid arguments to **toggle**.

autoflush If **autoflush** and **localchars** are both **true** and the AO, INTR, and QUIT characters are recognized and transformed into TELNET sequences, **telnet** does not display any data on the user's terminal until the remote system acknowledges (with a TELNET **timing mark** option) that it has processed those TELNET sequences. The initial value of **autoflush** is **true** if the terminal has not done an **stty noflsh**, and **false** if it has.

autosynch If **autosynch** and **localchars** are both **true**, then typing the INTR or QUIT character sends that character's TELNET sequence, followed by the TELNET SYNC sequence. This procedure causes the remote host to discard all previously typed input until both of the TELNET sequences have been read and acted upon. The initial value of this toggle is **false**.

crmod Toggles carriage return mode. When set to **true**, most carriage return characters received from the remote host are mapped into a carriage return followed by a line feed. This mode does not affect the characters typed by the user, only those received from the remote host. This mode is useful when the remote host sends only a carriage return and not a line feed. The initial value of this toggle is **false**.

debug Toggles debugging at the socket level. This argument can only be entered by a user with superuser privileges. The initial value of this toggle is **false**.

localchars Determines the handling of TELNET special characters. When this value is **true**, the **ERASE**, **FLUSH**, **INTERRUPT**, **KILL**, and **QUIT** characters are recognized locally and transformed into the appropriate TELNET control sequences (EC, AO, IP, BRK, and EL, respectively). When this value is **false**, these special characters are sent to the remote host as literal characters. The initial value of **localchars** is **true** in line-by-line mode and **false** in character-at-a-time mode.

netdata Toggles the display of all network data (in hexadecimal format). The data is written to standard output unless a *net_trace_file* is specified with the **-n** flag on the **telnet** command line. The initial value of this toggle is **false**.

options Toggles the display of internal Telnet Protocol processing options such as terminal negotiation and local or remote echo of characters. The initial value of this toggle is **false**, indicating that the current options should not be displayed.

z Opens a shell on the local host. The shell started is the one specified by the **SHELL** environment variable. When the shell is exited using the local **END OF FILE** sequence (**Ctrl-d** by default), **telnet** returns to the remote session.

Examples

Note: In the following examples, if you enter **tn** instead of **telnet**, the command mode prompts appears as `tn>`.

1. To log in to `host1` and do terminal type negotiation:

```
$ telnet host1
Trying . . .
Connected to host1
Escape character is ^T

AIX telnet (host1)
IBM AIX/RT Operating System
Version 2.2.1 (C) COPYRIGHT IBM CORP. 1985, 1988
(/dev/pts0)
login: _
```

2. To log in to hostl as a **vt100** (no terminal type negotiation), choose one of the following methods:

   a. Use the following commands to set the **EMULATE** environment variable for this login session, then enter **telnet**.

```
$ EMULATE=vt100; export EMULATE
$ telnet hostl
Trying . . .
Connected to hostl
Escape character is ^T

AIX telnet (hostl)
IBM AIX/RT Operating System
Version 2.2.1 (C) COPYRIGHT IBM CORP. 1985, 1988
(/dev/pts0)
login: _
```

   b. Use the following command to set the terminal type for this **telnet** session only.

```
$ telnet -e vt100 hostl
Trying . . .
Connected to hostl
Escape character is ^T

AIX telnet (hostl)
IBM AIX/RT Operating System
Version 2.2.1 (C) COPYRIGHT IBM CORP. 1985, 1988
(/dev/pts0)
login: _
```

3. To log in to remote host host3 as user systest, and then check the status of the **telnet** program:

```
$ telnet host3
Trying . . .
Connected to host3
Escape character is ^T

AIX telnet (host3)
IBM AIX/RT Operating System
Version 2.2.1 (C) COPYRIGHT IBM CORP. 1985, 1988
(/dev/pts0)
login: systest
Password: <Enter password>
$ ^t
telent> status
Connected to host3.
Operating in character-at-a-time mode.
Escape character is '^T'.
_ <Press Enter>
$_
```

Upon completion of the **status** subocmmand, you must press **Enter** to return to the remote prompt.

Files

$HOME/.3270keys	Defines user's 3270 keyboard mapping.
/etc/3270.keys.rt	Defines standard RT 3270 keyboard (console) mapping.
/etc/3270.keys	Defines base 3270 keyboard mapping for use with limited function terminals.

Related Information

In this book: "**telnetd**" on page 3-41.

The **pty** device driver in *AIX Operating System Technical Reference.*

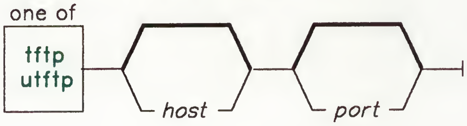
tftp, utftp

Purpose

Provides the Trivial File Transfer Protocol.

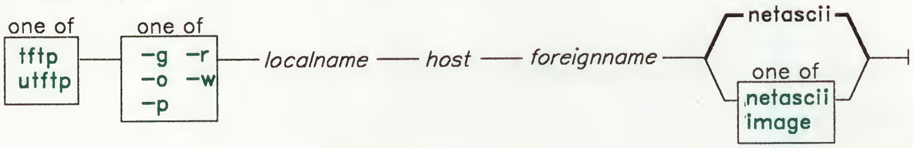
Syntax

Interactive Form



A5ACC041

Command Line Form



A5ACC013

Description

The **tftp** and **utftp** commands transfer files between hosts using the Trivial File Transfer Protocol (TFTP). Since TFTP is a minimal file transfer protocol, the **tftp** and **utftp** commands do not provide all of the features described under "**ftp**, **xftp**" on page 2-9. For example, **tftp** and **utftp** do not provide commands to list remote files or change directories at the foreign host, and limited file access privileges are given to the remote TFTP server.

The remote host must have a **tftpd** server started by its **inetd** server and have an account defined that limits the access of **tftpd**. For more information on setting up this account for your host, see "**tftpd**" on page 3-43.

Note: The **tftp** and **utftp** commands are not available when your host is operating in controlled access mode.

The **tftp** and **utftp** commands have two forms: ***interactive*** form and ***command line*** form. In the interactive form, **tftp** and **utftp** are issued alone or with a *host* that specifies the default host to use for file transfers during this session. If you choose, you can also specify the *port* the **tftp** or **utftp** connection should use, such as the one specified for **mail** in the **/etc/services** file. When you enter the interactive form of either of these commands, the `tftp>` prompt appears. In the command line form, all of the arguments to either command are specified on the command line, and no prompt appears.

The command line forms of **tftp** and **utftp** are equivalent, except that **utftp** does not overwrite a local file. **tftp** can overwrite a file, but prompts the user before doing so. Because it is not interactive, the command line form of **utftp** can be more useful than **tftp** in a pipe.

When transferring data to a foreign host, the transferred data is placed in the directory specified by the *foreignname* parameter. *foreignname* must be a fully specified file name, and the remote file must both exist and have write permission set for others. **tftp** attempts to write the data to the specified file. However, if the remote TFTP server does not have the appropriate privileges to write the remote file or if the file does not already exist, the transfer fails.

Interactive Form

Once the `tftp>` prompt appears, the following commands can be issued:

? [*command*]
: Displays help information. If a *command* is specified, only information about that command is displayed.

ascii
: Synonym for the **mode ascii** command.

binary
: Synonym for the **mode binary** command.

connect *host* [*port*]
: Sets the remote host, and optionally the *port*, for file transfers. Since the TFTP protocol does not maintain connections between transfers, the **connect** command does not create a connection to the *host*, but stores it for transfer operations. Because the remote host can be specified as part of the **get** or **put** commands, which overrides any host previously specified, the **connect** command is not required.

get *remfile* [*locfile*]
get *remfile remfile remfile* [*remfile* . . .]
: Gets a file or set of files from the remote host to the local host. Each of the *remfile* parameters can be specified in one of the following two ways:

- As a *file* that exists on the remote host if a default host has already been specified, or

- As *host:file*, where *host* is the remote host and *file* is the name of the file to copy to the local system. If this form of the parameter is used, the last *host* specified becomes the default host for later transfers in this **tftp** session.

mode *type*
Sets the *type* of transfer mode to that specified, either **ascii** or **binary**. A transfer mode of **ascii** is the default.

put *locfile* [*remfile*]
put *locfile locfile locfile* [*locfile* . . .] *remotedir*
Puts a file or set of files from the local host onto the remote host. The *remotedir* and *remfile* parameters can be specified in one of the following two ways:

- As a file or directory that exists on the remote host if a default host has already been specified, or
- As *host:remfile*, where *host* is the remote host and *remfile* is the name of the file or directory on the remote system. If this form of the parameter is used, the last *host* specified becomes the default host for later transfers in this **tftp** session.

In either of these cases, the remote file or directory name must be a fully specified path name, even if the local and remote directories have the same name. If a remote directory is specified, the remote host is assumed to be a UNIX machine.

quit
Exits **tftp**. An **END OF FILE** also exits the program.

rexmt *value*
Defines the retransmission timeout for each packet, in seconds.

status
Shows the current status of **tftp**, including the current transfer mode (**ascii** or **binary**), connection status, timeout value, and so on.

timeout *value*
Sets the total transmission timeout to the number of seconds specified by *value*.

trace
Turns packet tracing on or off.

verbose
Turns verbose mode, which displays additional information during file transfer, on or off.

Command Line Form

In this form, if the *action* flag is:

-w *or* **-p** Writes (or puts) local data, specified by *localname*, to the file *foreignname* on the foreign host specified by *host*. If *localname* is a file name, **tftp** transfers the specified local file. If *localname* is specified as a dash (**-**), **tftp** transfers data from local standard input to the foreign host. When *localname* is standard input, **tftp** allows 25 seconds for all input to be entered before timing out.

-r *or* **-g** *or* **-o** Reads (or gets) remote data from the file *foreignname* at the foreign host specified by *host* and writes it to *localname*. If *localname* is a file name, **tftp** writes the data to the specified local file. For the **-r** and **-g** actions, **tftp** prompts for verification before overwriting an existing local file. For the **-o** action, **tftp** overwrites an existing local file without prompting. If *localname* is specified as a dash (**-**), **tftp** writes the data to local standard output.

Note: Since **tftp -g** or **tftp -r** prompt before overwriting an existing local file, it may be impractical to use **tftp** in a pipe. The **utftp** command performs the same **-r** and **-g** actions as **tftp**, but simply stops before overwriting a local file. Thus, **utftp** may be more appropriate for use in a pipe.

For both of the following modes of file transfer, the *foreignname* parameter is the name of a file that has write permission set for others. Note that *foreignname* must be in double quotes (" ") if it contains shell special characters.

The mode of transfer is one of the following:

netascii Transfers the data as 7-bit ASCII characters in 8-bit transfer bytes. This is the default.

image Transfers the data as 8-bit binary data bytes in 8-bit transfer bytes, with no conversion. **image** transfer can be more efficient than **netascii** transfer when transferring between two RT hosts. **netascii** should be used when transferring ASCII files from an RT system to a different type of host.

Examples of Command Line Form

1. To transfer a binary file, `core`, in the current directory to the directory `/tmp` on host `host3`:

```
$ tftp -w core host3 /tmp/core image
Sent 309295 bytes in 15 seconds
$ _
```

2. To copy the /etc/hosts file from host host3 and redirect the output to standard output of the local host:

```
$ tftp -g - host3 /etc/hosts
192.100.13.3 nameserver
192.100.13.3 host2
192.100.13.5 host1
192.100.13.7 host3
192.100.13.3 timeserver
Received 128 bytes in 0.4 seconds
$ _
```

3. The following command gets the file /u/john/schedule from the foreign host host1, and writes it into the local file newsched:

```
$ utftp -g newsched host1 /u/john/schedule
$ _
```

4. The following command gets the file /u/john/schedule from the foreign host host1, pipes it to the **grep** command, and writes the results to the local file jones.todo:

```
$ utftp -g - host1 /u/john/schedule | grep Jones > jones.todo
$ _
```

Example of Interactive Form

1. To enter **tftp**, check the current status, connect to host1, and transfer the file update from host1 to the local host:

```
$ tftp
tftp> status
Not connected.
Mode: netascii   Verbose: off   Tracing: off
Rexmt_interval: 5 seconds, Max-timeout: 25 seconds
tftp> connect host1
tftp> get /u/alice/update update
```

Note: Directory **/u/alice** on the remote host must have read permission set for others.

Related Information

In this book: "**tftpd**" on page 3-43.

tn

Purpose

Provides the TELNET interface for logging in to a foreign host.

Related Information

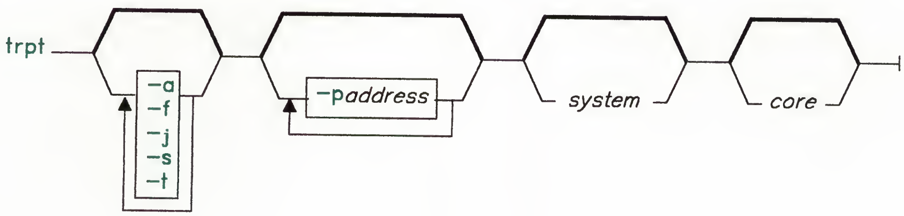
For a full description of this command, see "**telnet**, **tn**" on page 2-72.

trpt

Purpose

Performs protocol tracing on sockets.

Syntax



A5ACC043

Description

The **trpt** command queries the buffer of TCP trace records created when a socket is marked for *debugging* with the **setsockopt** routine. (For more information on setting socket options, see the **setsockopt** routine in the *AIX Operating System Technical Reference.*) **trpt** then prints a description of these trace records.

When no options are used, **trpt** prints all the trace records found in the system, grouped according to TCP connection protocol control block (PCB).

If debugging is being performed on a system or core file other than the default, the *system* and *core* arguments can be used to replace the defaults.

The **trpt** command can be used in the following way:

1. Isolate the problem and enable debugging on the socket or sockets involved in the connection.

2. Find the address of the protocol control blocks associated with these sockets by using the **netstat -aA** command.

3. Run **trpt**, using the **-p** flag to supply the associated protocol control block addresses. You can specify multiple **-p** *address* flags with a single **trpt** command.

The **-f** flag to **trpt** can be used to follow the trace log once it is located, while the **-j** flag can be useful in checking the presence of trace records for the socket in question.

If the system image does not contain the proper symbols to find the trace buffer, the **trpt** command fails.

Flags

-a Prints the values of the source and destination addresses for each packet recorded, in addition to the normal output.

-f Follows the trace as it occurs, waiting briefly for additional records each time the end of the log is reached.

-j Lists just the protocol control block addresses for which there are trace records.

-p *address* Shows only trace records associated with the protocol control block specified in hexadecimal by the *address* parameter. You must include a space between the **-p** and the *address* when you specify this flag.

-s Prints a detailed description of the packet sequencing information, in addition to the normal output.

-t Prints the values for all timers at each point in the trace, in addition to the normal output.

Examples

1. To get a list of the protocol control blocks that have trace records, enter:

```
$ trpt -j
124b0c, 12500c
$ _
```

2. To print out the trace records associated with a single protocol control block:

```
$ trpt -p 12500c
800 ESTABLISHED:output ad8eaa13@2326e6e5(win=4000)<ACK> -> ESTABLISH
ED
800 ESTABLISHED:input [2326e6e5..2326e727)@ad8eaa13(win=1ef)<ACK,
PUSH> -> ESTABLISHED
800 ESTABLISHED:user RCVD -> ESTABLISHED
900 ESTABLISHED:output ad8eaa13@2326e727(win=4000)<ACK> -> ESTABLISH
```

```
ED
900 ESTABLISHED:input [2326e727..2326e82f)@ad8eaa13(win=1ef)<ACK,
PUSH> -> ESTABLISHED
900 ESTABLISHED:user RCVD -> ESTABLISHED
900 ESTABLISHED:output ad8eaa13@2326e82f(win=4000)<ACK> -> ESTABLISH
ED
900 ESTABLISHED:input 2326e82f@ad8eaa13(win=1ef)<ACK,FIN,PUSH> -> CL
OSE_WAIT
900 CLOSE_WAIT:output ad8eaa13@2326e830(win=4000)<ACK> -> CLOSE_WAIT
900 LAST_ACK:output ad8eaa13@2326e830(win=4000)<ACK,FIN> -> LAST_ACK
900 CLOSE_WAIT:user DISCONNECT -> LAST_ACK
900 LAST_ACK:user DETACH -> LAST_ACK
$ _
```

3. To print out trace information and the source and destination addresses for each packet recorded, enter:

```
$ trpt -a
124b0c:
900 ESTABLISHED:input (src=192.9.201.3,4257, dst=192.9.201.2,102
5)2326e6e5@ad938c02(win=200)<ACK,FIN,PUSH> -> CLOSE_WAIT
900 CLOSE_WAIT:output (src=192.9.201.2,1025, dst=192.9.201.3,425
7)ad938c02@2326e6e6(win=4000)<ACK> -> CLOSE_WAIT
900 LAST_ACK:output (src=192.9.201.2,1025, dst=192.9.201.3,4257)
ad938c02@2326e6e6(win=4000)<ACK,FIN> -> LAST_ACK
900 CLOSE_WAIT:user DISCONNECT -> LAST_ACK
900 LAST_ACK:user DETACH -> LAST_ACK

12500c:
800 ESTABLISHED:output (src=192.9.201.2,1024, dst=192.9.201.3,51
2)ad8eaa13@2326e6e5(win=4000)<ACK> -> ESTABLISHED
800 ESTABLISHED:input (src=192.9.201.3,512, dst=192.9.201.2,1024)
[2326e6e5..2326e727)@ad8eaa13(win=1ef)<ACK,PUSH> -> ESTABLISHED
800 ESTABLISHED:user RCVD -> ESTABLISHED
900 ESTABLISHED:output (src=192.9.201.2,1024, dst=192.9.201.3,51
2)ad8eaa13@2326e727(win=4000)<ACK> -> ESTABLISHED
900 ESTABLISHED:input (src=192.9.201.3,512, dst=192.9.201.2,1024)
[2326e727..2326e82f)@ad8eaa13(win=1ef)<ACK,PUSH> -> ESTABLISHED
900 ESTABLISHED:user RCVD -> ESTABLISHED
```

```
900 ESTABLISHED:output (src=192.9.201.2,1024, dst=192.9.201.3,51
2)ad8eaa13@2326e82f(win=4000)<ACK> -> ESTABLISHED
900 ESTABLISHED:input (src=192.9.201.3,512, dst=192.9.201.2,1024)
2326e82f@ad8eaa13(win=1ef)<ACK,FIN,PUSH> -> CLOSE_WAIT
900 CLOSE_WAIT:output (src=192.9.201.2,1024, dst=192.9.201.3,512)
ad8eaa13@2326e830(win=4000)<ACK> -> CLOSE_WAIT
900 LAST_ACK:output (src=192.9.201.2,1024, dst=192.9.201.3,512)a
d8eaa13@2326e830(win=4000)<ACK,FIN> -> LAST_ACK
900 CLOSE_WAIT:user DISCONNECT -> LAST_ACK
900 LAST_ACK:user DETACH -> LAST_ACK
$ _
```

Related Information

In this book: "**netstat**" on page 2-39.

The **setsockopt** routine in *AIX Operating System Technical Reference.*

xftp

Purpose

Transfers files between a local and a remote host.

Related Information

For a full description of this command, see "**ftp**, **xftp**" on page 2-9.

Chapter 3. Server Commands

CONTENTS

About This Chapter

This chapter describes the server commands, or daemons, included with the IBM RT Interface Program for use with TCP/IP. Server commands provide support for user commands (described in Chapter 2, "User Commands" on page 2-1). Server commands usually are started either at system start by entries in the **/etc/rc.tcpip** file or by the **inetd** command as needed. Server commands usually are not entered on the command line.

The following is a list of the server commands, grouped according to function:

- File transfer

  ftpd
  tftpd
  uucpd[1]

- Remote mail and conversations

  sendmail[1]
  talkd

- Remote login, command execution, and printing

  lpd
  rexecd
  rshd
  rlogind
  telnetd

- Network management

  fingerd
  gated
  inetd
  named
  portmap[1]
  routed
  rwhod
  syslogd[1]

In this chapter, the server commands are organized alphabetically.

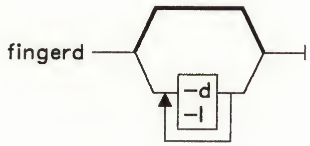
1 These commands are not part of the Interface Program. For a description, refer to *AIX Operating System Commands Reference.*

fingerd

Purpose

Provides the server function for the **finger** command.

Syntax



A5ACC015

Description

The **fingerd** command is the server for the Name/Finger Protocol (FINGER) used by the **finger** command. **fingerd** uses the TCP protocol. Since the **fingerd** server is normally started by the **inetd** server, the **fingerd** command should not be issued from the command line.

When **fingerd** receives a FINGER request from a foreign host, it returns information about a particular user at the host, or about all users logged in at the host, to the foreign host.

The **fingerd** server should have a user ID with the fewest privileges possible, which is by convention called **nobody**. To allow use of the **fingerd** server on your host, create a user ID with a separate group that has very restricted privileges. If you decide not to call this user ID **nobody**, edit the **/etc/inetd.conf** file and change the `nobody` on the line that begins with `finger` to the user name you selected. In addition, whether you decide to change the name of the **fingerd** user or not, you must edit the **/etc/inetd.conf** file and remove the comment symbol (#) from the beginning of the `finger` line. This allows the **fingerd** server to be started when needed by the **inetd** server.

Flags

-d Sends debugging information to the **syslogd**. For further information on the files used by this daemon, see **syslogd** in *AIX Operating System Commands Reference.*

-l Sends logging information to the **syslogd** about who is connected. For further information on the files used by this daemon, see **syslogd** in *AIX Operating System Commands Reference.*

Files

/etc/passwd	User database.
/etc/security/passwd	Password file.
/etc/utmp	**who** file.
$HOME/.plan	Plans for requested user.
$HOME/.project	Projects for requested user.
/usr/bin/whois	**whois** command.
/bin/who	**who** command.

Related Information

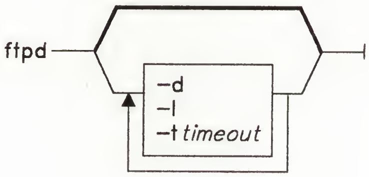
In this book: "**finger**" on page 2-6.

ftpd

Purpose

Provides the server function for the FTP protocol.

Syntax



A5ACC019

Description

The **ftpd** command is the server process for the File Transfer Protocol (FTP). **ftpd** uses the Transmission Control Protocol (TCP) and listens at the port specified with `ftp` in the **/etc/services** file. Since the **ftpd** server is normally started by the **inetd** server, the **ftpd** command should not be issued from the command line.

ftpd expands file names according to the conventions of the **csh** command. This allows using metacharacters like asterisk (*), question mark (?), left and right brackets ([]), left and right braces ({}), and tilde (~).

ftpd authenticates users according to these rules:

- The user name must be in the password database, **/etc/passwd**, and not have a null password. (In this case, the client process must provide the password before any file operations are performed.)
- The user name must not be in the **/etc/ftpusers** file.
- The user must have a standard shell returned by the **getusershell** routine.

If the user name is `anonymous` or `ftp`, an anonymous `ftp` account must be defined in the password file. In this case, the client process is allowed to log in using any password. By convention, the password is the name of the client host. **ftpd** takes special measures to restrict access by the client process of the anonymous account.

When handling an anonymous FTP user, the server performs a **chroot** to the home directory of the `ftp` user account. To ensure that system security is not breached, it is recommended that the home directory for the `ftp` account be constructed with care. An example shell script called **anon.ftp** is provided in the **/usr/lpp/tcpip/samples** directory to demonstrate a possible structure of the `ftp` subtree. Further information on setting up the anonymous ftp account is available in that file.

Flags

-d Sends debugging information to the **syslogd**. For further information on the files used by this daemon, see **syslogd** in *AIX Operating System Commands Reference.*

-l Sends logging information to the **syslogd**. For further information on the files used by this daemon, see **syslogd** in *AIX Operating System Commands Reference.*

-t*timeout* Logs out inactive sessions after *timeout* minutes.

Requests

The **ftpd** server supports the following FTP requests:

ABOR Aborts transaction.

ACCT No operation.

ALLO No operation.

APPE Appends to a file.

CDUP Changes to the parent directory.

CWD Changes current directory.

DELE Deletes a file.

HELP Gives help information.

LIST Gives list files in a directory (`ls -lg`).

MKD Creates a directory.

MODE Specifies data transfer *mode.*

NLST Gives name list of files in directory (`ls`).

NOOP Does nothing.

PASS Specifies password.

PASV	Prepares for server-to-server transfers.
PORT	Specifies data connection port.
PWD	Prints the current working directory.
QUIT	Terminates session.
RETR	Retrieves a file.
RMD	Removes a directory.
RNFR	Specifies rename-from file name.
RNTO	Specifies rename-to file name.
STOR	Stores a file.
STOU	Stores a file using a unique file name.
STRU	Specifies data transfer *structure*.
TYPE	Specifies data transfer *type*.
USER	Specifies user name.
XCUP	Changes to the parent directory.
XCWD	Changes current directory.
XMKD	Creates a directory.
XPWD	Prints the current working directory.
XRMD	Removes a directory.

The remaining FTP requests defined in Internet RFC 959 are recognized but not acted on.

Files

/etc/locks/ftpd	Interlock, PID storage.
/etc/passwd	User database.
/etc/security/passwd	Password file.
/usr/lpp/tcpip/samples/anon.ftp	Example shell script to set up anonymous **ftp** account.

Related Information

In this book: "**ftp**, **xftp**" on page 2-9.

The **/etc/passwd** and **/etc/security/passwd** files in *AIX Operating System Technical Reference.*

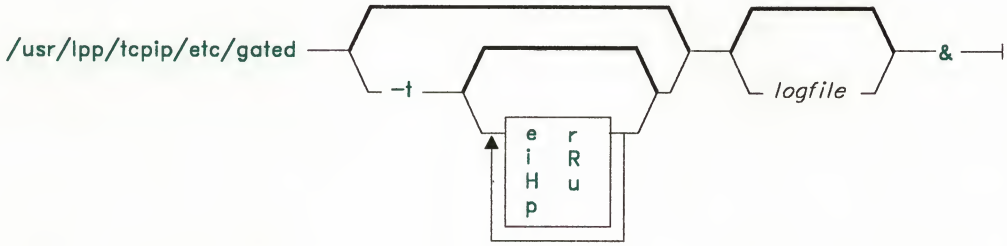
The **syslogd** command in *AIX Operating System Commands Reference.*

gated

Purpose

Provides gateway routing functions.

Syntax



A5ACC045

Description

The **gated** server is a routing daemon that handles multiple routing protocols, including the RIP, EGP, and HELLO routing protocols. The **gated** process can be configured to perform all of these routing protocols or any combination of the three. The configuration file for **gated** is **/etc/gated.conf**, and **gated** stores its process ID in the file **/etc/gated.pid**.

Either **gated** or **routed** can be run on a gateway host. If EGP or HELLO routing is needed, use **gated**. If only RIP routing is needed, use either **gated** or **routed**. (See “**routed**” on page 3-28 for more information on the function of **routed**.)

Note: The **gated** and **routed** daemons should not both be run on the same host, as this may produce unpredictable results.

When using **gated**, all protocols have a 2-minute hold down. When a routing update indicates that the route in use is being deleted, **gated** does not delete the route for 2 minutes.

The **gated** server can be started with a log file, which is specified on the command line. One or more trace flags can also be entered on the command line or specified in the configuration file in the **traceflags** stanza. **gated** forks and detaches itself from the controlling terminal unless trace flags are specified without a log file. In this case, all trace output is sent to the controlling terminal.

If routing restrictions are used, **gated** uses the **syslogd** program at log level **LOG_WARNING** and facility **LOG_DAEMON** to record all invalid networks. (For further information on the **syslogd** command, including how to change the log facility, see *AIX Operating System Commands Reference.*) Given the complexity of the Internet network, routing restrictions are useful in preventing access by unauthorized networks. While using routing restrictions requires a little more maintenance time, it is necessary to ensure the security of the Internet.

If EGP is being used when supplying the default route (via RIP gateway or HELLO gateway) and all EGP neighbors are lost, the default route is not advertised until at least one EGP neighbor is regained.

RIP propagates and listens to host routes. This allows **gated** to handle point-to-point links more consistently. **gated** also supports the RIP_TRACE commands.

The **gated** process detects changes made to the interfaces without having to restart the process. If the net mask, subnet mask, broadcast address, or interface metric is changed, the interface should be marked down with the **ifconfig** command, and then marked up at least 30 seconds later. Flag changes do not require the interface to be brought down and back up.

Subnet interfaces are supported. Subnet information is only propagated on interfaces to other subnets of the same network. For example, if there is a gateway between two class B networks, the subnet routes for each respective class B net are not propagated into the other class B net. Only the class B network number is propagated.

gated listens to host and network REDIRECTs, and tries to take an action on the REDIRECT for its own internal tables that parallels the kernel's action. In addition, **gated** times out routes learned via a REDIRECT after 6 minutes. The route is then deleted from the kernel routing tables. This helps keep the routing tables consistent. Routes that are learned by REDIRECTs are not announced by any routing protocol.

The **gated** EGP code verifies that all nets sent and received are valid class A, B, or C networks per the EGP specification. Information about networks that do not meet these criteria is not propagated. If an EGP update packet contains information about a network that is not class A, B, or C, the update is considered to be in error and is ignored.

Internal Metrics for gated

gated stores all metrics internally as a time delay in milliseconds to preserve the granularity of HELLO time delays. The internal delay ranges from 0 to 30000 milliseconds, with 30000 representing infinity. Metrics from other protocols are translated to and from a time delay as they are received and transmitted. EGP distances are not comparable to HELLO and RIP metrics but are stored as time delays internally for comparison with other EGP metrics. The conversion factor between EGP distances and time delays is 100. RIP and interface metrics are translated to and from the internal time delays with the use of the following translation tables. The first two columns represent the time delay to RIP metric translation, while the second two columns represent the RIP metric to time delay translation.

Time Delay (milliseconds)			RIP Metric
0	to	0	0
1	to	100	1
101	to	148	2
149	to	219	3
220	to	325	4
326	to	481	5
482	to	713	6
714	to	1057	7
1058	to	1567	8
1568	to	2322	9
2323	to	3440	10
3441	to	5097	11
5098	to	7552	12
7553	to	11190	13
11191	to	16579	14
16580	to	24564	15
24565	to	30000	16

RIP Metric	Time Delay (milliseconds)
0	0
1	100
2	148
3	219
4	325
5	481
6	713
7	1057
8	1567
9	2322
10	3440
11	5097
12	7552
13	11190
14	16579
15	24564
16	30000

Flags

The valid trace flags are as follows:

-t If used alone, logs all error messages, route changes, and EGP packets sent and received. Using this flag alone starts the **i**, **e**, **r**, and **p** trace flags as well. When used with another flag, the **-t** flag has no effect and only the accompanying flags are recognized. Note that when using any other flags, they must be preceded by the **-t** flag.

i Logs all internal errors and interior routing errors.

e Logs all external errors due to EGP, exterior routing errors, and EGP state changes.

r Logs all routing changes.

p Traces all EGP packets sent and received.

u Logs all routing updates sent.

R Traces all RIP packets received.

H Traces all HELLO packets received.

The **gated** daemon always logs fatal errors. If no log file is specified and none of the preceding trace flags are set, all messages are sent to **/dev/null**.

Signals

The **gated** server performs the following actions when sent the **SIGHUP** and **SIGINT** signals using the **kill** command.

When a **SIGHUP** signal is sent to the **gated** process and **gated** was invoked with both trace flags and a log file, tracing is toggled off and the log file is closed. At this point the log file can be moved or deleted. The next **SIGHUP** to **gated** toggles tracing on. **gated** reads the **/etc/gated.conf** configuration file and sets the trace flags to those specified by the **traceflags** stanza. If no **traceflags** stanza exists, tracing resumes using any trace flags that were specified on the command line. Trace output is sent to the log file specified on the command line. The output is appended if the log file already exists, and the file is created if it does not exist. This is useful for having rotating log files similar to those of **syslogd**.

Sending **gated** a **SIGINT** signal causes a memory dump to be scheduled within the next 60 seconds. The memory dump is written to the file **/usr/tmp/gated_dump**. **gated** processes all pending routing updates before performing the memory dump. The memory dump contains a snapshot of the current **gated** status, including the interface configurations, EGP neighbor status, and the routing tables. If the file **/usr/tmp/gated_dump** already exists, the memory dump is appended to the existing file.

Files

/etc/gated.conf	Contains **gated** configuration information.
/etc/gated.pid	Contains the **gated** process ID.
/usr/tmp/gated_dump	Specifies the memory dump file.

Related Information

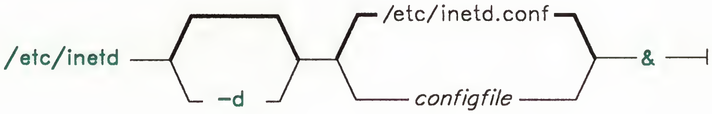
In this book: "**routed**" on page 3-28 and "**gated.conf**" on page 4-7.

inetd

Purpose

Provides Internet service management.

Syntax



A5ACC033

Description

The **inetd** command is the *super server* for handling some Internet services. Essentially, **inetd** is designed to reduce system load by only running one daemon, **inetd**, to invoke other daemons if, and only if, they are needed. Additionally, **inetd** reduces system load by providing several simple Internet services internally without invoking other daemons.

inetd should be started at boot time by an entry in the **/etc/rc.tcpip** file. Once started, **inetd** listens for connections on certain Internet sockets specified in the **/etc/services** file. When **inetd** receives a request on one of these sockets, it determines what service corresponds to that socket and either handles the service request itself or invokes the appropriate server.

When **inetd** starts, it reads its configuration information from the configuration file *configfile*, if specified, or from **/etc/inetd.conf**, otherwise. This file describes to **inetd** how Internet service requests on Internet sockets should be handled. The entries in this file include the following information:

Service Name The name of a valid service. (Valid services are defined in the **/etc/services** file.)

Socket Type The type of Internet socket used for the service. (Only stream and datagram sockets are implemented.)

Protocol The Internet protocol used for the service. (Valid protocols are defined in the **/etc/protocols** file.)

Wait/Nowait Whether **inetd** should wait for the service to complete before continuing to listen for this type of service request.

User The user name to be used by the server.

Server The server to be invoked to handle the service. (The fully specified file name of an invoked server or, if **inetd** handles the service internally, `internal`.)

Arguments The command line used to start the server if **inetd** does not handle the service internally.

See "**inetd.conf**" on page 4-32 for a detailed description of an **inetd** configuration file.

Flag

-d Sends debugging messages to the **syslogd**. For further information on the files used by this daemon, see **syslogd** in *AIX Operating System Commands Reference.*

Signal

The following signal has the specified effect when sent to the **inetd** process using the **kill** command.

SIGHUP Rereads the **inetd** configuration file.

Requests

The Internet service requests supported internally by **inetd** are generally used for debugging. They include:

ECHO Returns data packets to a client host.

DISCARD Discards received data packets.

CHARGEN Discards received data packets and sends predefined or random data.

DAYTIME Sends the current date and time in human-readable form.

TIME Sends the current date and time in machine-readable form.

Files

/etc/inetd.conf Default configuration file for **inetd**.

/etc/protocols Defines the names and numbers of valid Internet protocols.

/etc/rc.tcpip Boot file for starting TCP/IP daemons.

/etc/services Defines valid Internet services and their related sockets.

Related Information

In this book: "**ftpd**" on page 3-5, "**rlogind**" on page 3-26, "**rshd**" on page 3-33, "**talkd**" on page 3-39, and "**inetd.conf**" on page 4-32.

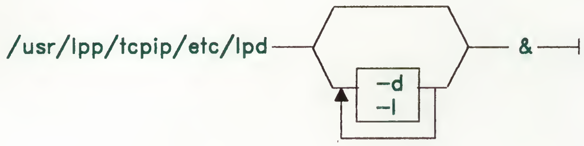
The **syslogd** command in *AIX Operating System Commands Reference.*

lpd

Purpose

Provides the server function for remote printing.

Syntax



A5ACC024

Description

The **lpd** command (the remote print server) monitors port 515 for print requests. Each request is placed in the **/usr/spool/lpd** directory. A print request consists of three parts:

- Control file
- Data
- Status file (AIX controls).

A host that can create a TCP/IP data stream and handle the **lpd** protocol can print remotely or act as a print server. As a security feature, **lpd** accepts print requests only from foreign hosts that are listed in the local **/etc/hosts.equiv** or **/etc/hosts.lpd** file.

The **lpd** daemon can run on any host in the network; its function is to accept print requests from foreign hosts (on port 515). **lpd** handles each request by forking a child process. Remote requests are first checked against the **/etc/hosts.equiv** and **/etc/hosts.lpd** files for permission to print on the local host.

Changes can be made to the **/etc/hosts.equiv** and **/etc/hosts.lpd** files without restarting the system. To put changes to these files into effect without restarting the system, send **SIGINT** (**kill -2** *PID*). Sending this signal causes the **/etc/hosts.equiv** and **/etc/hosts.lpd** database files to be reloaded.

The **lprbe** command creates three files in the spooling directory for each queued job. The names of these files are unique for each job, consisting of a special prefix for each type of file, a job sequence number, and the name of the originating host. The format for a spooling directory file name is:

spooling_directory/*x*fA#*host*

where:

x is one of the following prefixes:

c Control file

d Data file

t Temporary file.

is the sequence number of the job.

host is the name of the originating host.

The file name in the following example corresponds to the control file, sequence number `124`, from a job printed on the **lp** line printer queue on `host1`:

`/usr/spool/lpd/cfA124host1`

The three types of files created in the spool directory are:

Control A file that contains information about the job (for example, who the job is for, which host is to print the job, and what information should go on the title page). Each line of the control file is of the form **x***string*, where **x** is an element of the set shown in Figure 3-1.

Data A file that contains a copy of the file to be printed or a symbolic link to that file.

Temporary A file used to create the control and data files. This file is discarded once the job is queued.

Figure 3-1 lists the control codes processed by AIX:

C	**class name** on banner page
f	**file name**, text file to print
H	**host name** of host where **lprbe** was run
I	**indent**, amount to indent output, for **pr**
J	**job name** on banner page
L	**literal** name of user to print on banner

Figure 3-1 (Part 1 of 2). lpd Control File Codes

N name of file (used by **lpq**)
P **person**, the login ID of the user
p **file name**, text file to print with **pr**
T **title** for **pr**
U **unlink**, name of file to remove after printing
W **width**, width to print output, for **pr**

Figure 3-1 (Part 2 of 2). lpd Control File Codes

If the print request is sent by **lprbe** and **-naix** is not specified, AIX keywords of the form *-xxx=sss* are appended to the control file. These keywords include any hardcopy labels that have been defined by the system administrator for system where the print command was issued.

Following is a sample control file created when user `jim` prints the `/u/jim/foo.c` file from `host1`:

```
host1
Pjim
Jfoo.c
Chost1
Ljim
fdfA123host1
UdfA123host1
N/u/jim/foo.c
```

The local print client opens the known printer port, 515, on the remote host and sends the `\2printer\n` message, indicating to the service that it should prepare to receive a job. If the remote daemon does not respond with the message `\0`, there is some problem with the request; otherwise, the local daemon sends the control file to the remote daemon. Communication between the daemons is accomplished with lines of text if the first byte indicates what action is to be taken:

\1 Abandon the request; some problem prevents its completion.

\2 Read the control file.

\3 Read the data file.

The control file contains the names of the data files to be queued to print on the remote host. When the control file is sent, the local files are deleted. The remote **lpd** daemon starts a process to print the files. At this point, the local processing is finished.

Flags

-d Provides debugging information to the **syslogd**. For further information on the files used by this daemon, see **syslogd** in *AIX Operating System Commands Reference.*

-l Sends logging information to the **syslogd**. For further information on the files used by this daemon, see **syslogd** in *AIX Operating System Commands Reference.*

Examples

1. To start the **lpd** server daemon:

   ```
   # /usr/lpp/tcpip/etc/lpd &
   [process ID number]
   # _
   ```

2. To start the **lpd** server daemon with the debugging option:

   ```
   # /usr/lpp/tcpip/etc/lpd -d &
   [process ID number]
   # _
   ```

Files

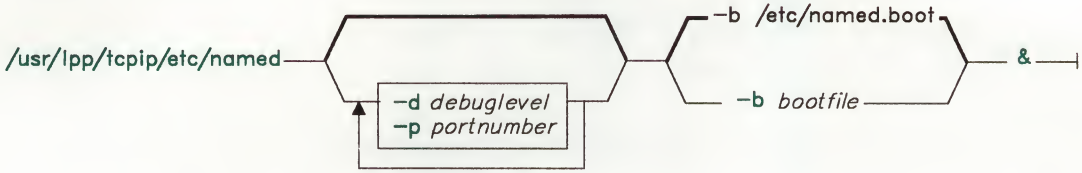
/dev/lp*	Print devices.
/etc/hosts.equiv	Names of hosts allowed to execute commands and print.
/etc/hosts.lpd	Names of hosts allowed to print only.
/usr/spool/lpd	Spool directory for control, status, and data files.

named

Purpose

Provides the server function for the Domain Name Protocol.

Syntax



A5ACC006

Description

The **named** daemon is the server for the Domain Name Protocol (DOMAIN). **named** is normally started at system boot time by an entry in the **/etc/rc.tcpip** command file. When **named** starts, it reads a boot file that describes local Internet domain information. This boot file is either *bootfile*, if specified, or **/etc/named.boot** otherwise. **named** uses the boot file (and data files referenced in the boot file) to set up the local domain database.

Note: The **named** data files referenced in the boot file must be in Standard Resource Record Format. Errors in the boot file or the data files may stop **named** from functioning properly. See "**named.boot**" on page 4-34 and "**named.***" on page 4-37 for more information on these files.

Once the database is set up, **named** begins listening for nameserver requests generated by resolver routines running on foreign hosts. The socket listened to is normally the socket defined in the **/etc/services** file with an entry beginning with `domain`. However, this socket assignment can be overridden using the `-p` flag on the command line. Local resolver routines directly access the domain database set up by **named**.

Note: Using **named** requires that the **/etc/resolv.conf** file exist. If this file exists, the local kernel and resolver routines assume that the DOMAIN protocol is to be used. If the **/etc/resolv.conf** file does not exist, the local kernel and resolver routines use the **/etc/hosts** database. **named** will not function properly if the local kernel and resolver routines are configured for the local database. On a DOMAIN nameserver host, the **/etc/resolv.conf** file must exist and be empty.

Flags

-d *debuglevel* — Provides a debugging option. The **-d** flag causes **named** to write debugging information to the file **/usr/tmp/named.run**. The *debuglevel* parameter determines the level of messages printed, with valid levels from 1 to 9.

-p *portnumber* — Reassigns the Internet socket that **named** listens at for DOMAIN requests. If this parameter is not specified, **named** listens at the socket defined in the **/etc/services** file with an entry that begins with `domain`.

Signals

The following signals have the specified effect when sent to the **named** process using the **kill** command:

SIGHUP	**named** reads the **/etc/named.boot** file and reloads the database.
SIGINT	**named** dumps the current database to **/usr/tmp/named_dump.db**.
SIGUSR1	Turns on debugging; each **SIGUSR1** increments the debugging level.
SIGUSR2	Turns off debugging.

Files

/etc/named.boot	Boot file for initializing **named** database.
/etc/named.pid	PID storage.
/etc/resolv.conf	Defines the local domain and name server for the kernel resolver routines.
/etc/services	Defines socket service assignments.
/usr/lpp/tcpip/samples/named.boot	Sample **named.boot** file.
/usr/lpp/tcpip/samples/named.data	Sample **named.data** file.
/usr/lpp/tcpip/samples/hosts.awk	Sample **awk** script for converting an **/etc/hosts** file to an **/etc/named.data** file.
/usr/lpp/tcpip/samples/addrs.awk	Sample **awk** script for converting an **/etc/hosts** file to an **/etc/named.rev** file.

Related Information

In this book: "**named.boot**" on page 4-34, "**named.***" on page 4-37, "**resolv.conf**" on page 4-64, and "**services**" on page 4-66.

The **kill** command in *AIX Operating System Commands Reference.*

portmap

Purpose

Provides mapping of RPC program numbers to port numbers on RPC servers.

Related Information

The **portmap** command in *AIX Operating System Commands Reference.*

rexecd

Purpose

Provides the server function for the **rexec** command.

Syntax

```
rexecd ——|
```

A5ACC023

Description

The **rexecd** command is the server for the **rexec** command; it processes commands issued by a foreign host and returns the output of those commands to the foreign host. **rexecd** sends and receives data over a TCP connection. Since the **rexecd** server is normally started by the **inetd** server, the **rexecd** command should not be issued from the command line.

When **rexecd** receives a request, it initiates the following protocol:

1. The server reads characters from the socket up to a null (\0) byte and interprets the resulting string as an ASCII number (decimal).
2. If the number received is non-zero, **rexecd** interprets it as the port number of a secondary stream to be used for standard error output. **rexecd** then creates a second connection to the specified port on the client machine.
3. A null terminated user name of up to 16 characters is retrieved on the initial socket.
4. A null terminated, encrypted password of up to 16 characters is retrieved on the initial socket.
5. A null terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.
6. **rexecd** validates the user (as is done at login) and, if successful, changes the user's home directory and establishes the user and group protections of the user. (If any part of this procedure fails, **rexecd** abandons its attempt to create the connection and returns a diagnostic message.)

7. A null byte is returned on the connection associated with standard error, and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by **rexecd**.

Related Information

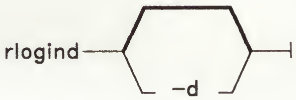
In this book: "**rexec**" on page 2-51.

rlogind

Purpose

Provides the server function for the **rlogin** command.

Syntax



A5ACC034

Description

The **rlogind** command is the server for the remote login command, **rlogin**. **rlogind** listens for service requests at the `login` socket as specified in the **/etc/services** file. Since the **rlogind** server is normally started by the **inetd** server, the **rlogind** command should not be issued from the command line.

When **rlogind** receives a service request, **rlogind** initiates the following protocol:

1. **rlogind** checks the source port number for the request. If the port number is not in the range 0-1023, **rlogind** terminates the connection.
2. **rlogind** uses the source address of the initial connection request to determine the name of the client host. If the name cannot be determined, **rlogind** uses the dot address of the client host.
3. **rlogind** allocates a pseudo-terminal and manipulates file descriptors so that the servant side of the pty becomes the standard input, standard output, and standard error for a local **login** process. (The **login** process is an instance of **login** invoked with a **-r** flag.)
4. The **login** process then proceeds with validating the client user, as described for **rshd**. If automatic login fails, the **login** process prompts the user to log in manually.

The parent of the **login** process manipulates the controller side of the pty, and operates as an intermediary between the local **login** process on the server host and the **rlogin** process on the client host. In normal operation, the packet protocol described for ptys is invoked to provide CTRL-S and CTRL-Q support and propagate interrupts to the client host. The **login** process propagates the client terminal type, as found in the environment variable

TERM. The screen or window size is requested from the client, and window size changes from the client are propagated to the pseudo-terminal.

All diagnostic messages are returned on the connection associated with standard error; after which, any network connections are closed. An error is indicated by a leading byte with a value of 1.

Note: Any pseudo-terminals (ptys) used by **rlogind** must be created with **logger** and **automatic enable** set to **false**.

Flag

-d Sends debugging messages to the **syslogd**. For further information on the files used by this daemon, see **syslogd** in *AIX Operating System Commands Reference.*

Files

/etc/hosts.equiv	Defines equivalent hosts.
$HOME/.rhosts	Defines equivalent remote users.
/etc/services	Defines Internet socket assignments.

Related Information

In this book: "**rlogin**" on page 2-54, "**rshd**" on page 3-33, and "**inetd**" on page 3-13.

The **pty** device driver in *AIX Operating System Technical Reference.*

The **login** command in *AIX Operating System Commands Reference.*

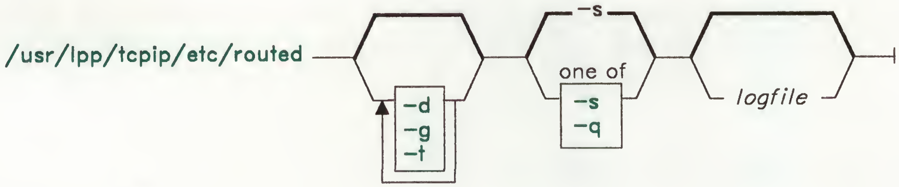
The **syslogd** command in *AIX Operating System Commands Reference.*

routed

Purpose

Manages network routing tables.

Syntax



A5ACC025

Description

The **routed** command is a daemon that manages the network routing tables in the kernel. **routed** uses a variant of the Xerox NS Routing Information Protocol (RIP)[2] to maintain current kernel routing table entries. **routed** is normally started at system start time by an entry in the **rc.tcpip** command file. **routed** uses the UDP protocol and listens for routing request, response, and update packets at socket 520.

Either **gated** or **routed** can be run on a gateway host. If EGP or HELLO routing is needed, use **gated**. If only RIP routing is needed, use either **gated** or **routed**. (See "**gated**" on page 3-9 for more information on the function of **gated**.)

Note: The **gated** and **routed** daemons should not both be run on the same host, as this may produce unpredictable results.

When **routed** starts, it finds any interfaces to directly connected hosts and networks that are configured into the system and marked as **up**. If multiple interfaces are present, **routed** assumes that the local host forwards packets between networks. **routed** transmits a RIP request packet on each interface (using a broadcast packet if the interface supports it) and then enters a loop, listening for RIP routing request and response packets from other hosts. If **routed** is to supply RIP information to other hosts, **routed** also

---

2 Defined in *Internet Transport Protocols*, XSIS 028112, Xerox System Integration Standard.

periodically sends RIP update packets (containing copies of its routing tables) to any directly connected hosts and networks.

When **routed** receives a RIP request packet and is to supply RIP routing information, **routed** generates a reply (response packet) based on the information maintained in the kernel routing tables. The response packet contains a list of known routes, each marked with a ***hop count metric*** (the number of host-to-host connections in the route). The metric for each route is relative to the sending host. A metric of 16 or greater is considered to be ***infinite***, or beyond reach.

routed uses information contained in RIP response and update packets from other hosts to update its routing tables. However, **routed** uses the information in a RIP routing packet to update the tables only if at least one of the following conditions exists:

- No routing table entry exists for the destination network or host, and the metric associated with the route is finite (that is, the metric is less than 16).
- The source host of the packet is the same as the router in the existing routing table entry. That is, updated routing information is being received from the same internetwork router through which packets for the destination are being routed.
- The existing entry in the routing table has not been updated for some time and the route is at least as efficient as the current route.
- The new route is shorter than the one to the same destination that is currently stored in the routing tables. (**routed** determines relative route length by comparing the new metric with the one stored in the routing table.)

When **routed** updates its internal routing tables, it generates a RIP update packet to all directly connected hosts and networks. Before updating the kernel routing tables, **routed** pauses for a brief period to allow any unstable conditions to stabilize.

Besides processing incoming RIP packets, **routed** also checks the internal routing table entries periodically. The metric for any entry that has not been updated for 3 minutes is set to infinity and marked for deletion. The deletion is delayed for 60 seconds so that information about the invalidated route can be distributed throughout the network. A host that acts as a RIP router supplies its routing tables to all directly connected hosts and networks every 30 seconds.

Hosts acting as internetwork routers send unsolicited routing tables every 30 seconds to all directly connected hosts and networks. The response is sent to the broadcast address on networks capable of that function, to the destination address on point-to-point links, and to the router's own address on other networks. The normal routing tables are bypassed when sending unsolicited responses. The reception of responses on each network is used to determine that the network and interface are functioning correctly. If no response is received on an interface, another route may be chosen to route around the interface, or the route may be dropped if no alternative is available.

Besides its ability to manage routes to directly connected hosts and networks, **routed** also employs the concept of ***distant*** gateways, both ***passive*** and ***active***, and ***external*** gateways.

These gateways cannot be identified via RIP queries. When it starts, **routed** reads the **/etc/gateways** file for information about these distant and external gateways.

The **/etc/gateways** file contains information about routes through distant gateways to hosts and networks that should be advertised via RIP, as well as external gateways. These routes can be either static routes to specific destinations or default routes to use when a static route to a destination is unknown.

If a gateway specified in the **/etc/gateways** file supplies RIP routing information, it should be marked **active**. ***Active gateways*** are treated as if they were network interfaces. That is, RIP routing information is distributed to the active gateway and, if no RIP routing information is received from the gateway for a period of time, **routed** deletes the associated route from the routing tables.

A gateway that does not exchange RIP routing information should be marked **passive**. ***Passive gateways*** are maintained in the routing tables indefinitely and information about them is included in any RIP routing information transmitted.

An **external** gateway is identified to inform **routed** that another routing process will install such a route and that **routed** should not install alternative routes to that destination. ***External gateways*** are not maintained in the routing tables and information about them is not included in any RIP routing information transmitted.

Note: Routes through external gateways must be to networks only.

For information about the **/etc/gateways** file, see "**gateways**" on page 4-21. To allow EGP routing, use the **gated** command. For information about EGP routing, see "**gated**" on page 3-9 and "**gated.conf**" on page 4-7.

If *logfile* is specified, **routed** writes information about its actions to the specified log file. The log contains information about any changes to the routing tables and a history of recent messages sent and received that are related to changed routes.

Flags

-d Enables additional debugging information, such as bad packets received, to be logged.

-g Causes the routing daemon to run on a gateway host. This flag is used on internetwork routers to offer a route to the default destination.

-q Prevents **routed** from supplying routing information whether it is functioning as an internetwork router or not. (**-q** is the opposite of **-s**; do not use these two flags together.)

-s Causes **routed** to supply routing information whether it is functioning as an internetwork router or not. (**-s** is the opposite of **-q**; do not use these two flags together.)

-t Causes all packets sent or received to be written to standard output. **routed** remains under control of the host that started it; therefore, an interrupt from the controlling host kills the **routed** process.

Signals

The following signals have the specified effect when sent to the **routed** process using the **kill** command:

SIGUSR1 Displays internal routing tables.

SIGHUP, SIGTERM, or **SIGQUIT**
Broadcasts RIP packets with hop counts set to *infinity*. Essentially, these signals disable the local host as a router. On a second **SIGHUP, SIGTERM,** or **SIGQUIT**, **routed** terminates.

Examples

1. To start **routed** (normally done automatically at system start) and cause it to return routing information whether or not it is functioning as an internetwork router, enter:

```
# /usr/lpp/tcpip/etc/routed -r -s &
[PID]
# _
```

2. To start **routed** without causing it to return routing information, enter:

```
# /usr/lpp/tcpip/etc/routed -q -r &
[PID]
# _
```

3. To start **routed** and cause it to write all packets sent or received to standard output, enter:

```
# /usr/lpp/tcpip/etc/routed -r -s -t &
[PID]
# _
```

4. To start **routed** and cause it to write its logging information to the file `routlog`, enter:

```
# /usr/lpp/tcpip/etc/routed -r -s -t routlog &
[PID]
# _
```

Files

/etc/gateways Routes through distant and external gateways.

/etc/networks Contains the network name database.

Related Information

In this book: "**route**" on page 2-57.

rshd

Purpose

Provides the server function for remote command execution.

Syntax

```
rshd ──┤
```

A5ACC035

Description

The **rshd** command is the server for the **rcp** and **rsh** commands. **rshd** provides remote execution of shell commands with user authentication based on requests from privileged sockets on trusted hosts. **rshd** listens at the socket defined in the **/etc/services** file with an entry beginning with `cmd`. Since the **rshd** server is normally started by the **inetd** server, the **rshd** command should not be issued from the command line.

When **rshd** receives a service request, it initiates the following protocol:

1. **rshd** checks the source port number for the request. If the port number is not in the range 0-1023, **rshd** terminates the connection.
2. **rshd** reads characters from the socket up to a null (\0) byte. The string read is interpreted as an ASCII number (base 10). If this number is non-zero, **rshd** interprets it as the port number of a secondary stream to be used as standard error. A second connection is created to the specified port on the client host The source port on the local host is also in the range 0-1023.
3. **rshd** uses the source address of the initial connection request to determine the name of the client host (see **gethostbyaddr** in *AIX Operating System Technical Reference*). If the name cannot be determined, **rshd** uses the dot address of the client host.
4. **rshd** retrieves the following information from the initial socket:
   a. A null-terminated string of at most 16 bytes interpreted as the user name of the user on the client host.
   b. A null-terminated string of at most 16 bytes interpreted as the user name to be used on the local server host.
   c. Another null-terminated string interpreted as a command line to be passed to a shell on the local server host.

5. **rshd** attempts to validate the user using the following steps:

   a. **rshd** looks up the local user name in the **/etc/passwd** file and attempts to execute a **chdir** to the home directory of the local user. If either the lookup or the **chdir** fail, **rshd** terminates the connection.

   b. If the local userid is not 0, **rshd** searches the **/etc/hosts.equiv** file to see if the name of the client host is listed. If the client host is listed as an equivalent host, **rshd** validates the user.

   c. If either the local userid is 0 or the client host is not an equivalent host, **rshd** checks if the local user's home directory contains a **.rhosts** file. If the file exists, **rshd** checks whether the remote host and user name are listed in the file. If the **.rhosts** file does exist and the remote host and user name are listed, **rshd** validates the user. Otherwise, **rshd** terminates the connection.

6. Once **rshd** validates the user, **rshd** returns a null byte on the initial connection and passes the command line to the user's local login shell. The shell then inherits the network connections established by **rshd**.

Files

/etc/hosts.equiv	Defines equivalent foreign hosts.
/etc/services	Defines Internet socket assignments.
$HOME/.rhosts	Defines equivalent remote users.

Related Information

In this book: "**rsh**, **remsh**" on page 2-60 and "**inetd**" on page 3-13.

The **pty** device driver and the **rcmd** routine in *AIX Operating System Technical Reference.*

rwhod

Purpose

Provides the server function for the network status.

Syntax

```
/usr/lpp/tcpip/etc/rwhod—— &——┤
```

A5ACC036

Description

The **rwhod** command is the server that maintains the database used by the **rwho** and **ruptime** commands. **rwhod** is normally started by an entry in the **/etc/rc.tcpip** command file. Once started, **rwhod** operates as both a producer and a consumer of status information.

As a producer, **rwhod** periodically (approximately every 3 minutes) queries the state of the local host and constructs status messages, which are broadcast to the network. As a consumer, **rwhod** listens for status messages from **rwhod** servers on foreign hosts. When **rwhod** receives a status message, it validates the received status message and records the message in the **/usr/spool/rwho** directory. (**rwho** and **ruptime** use the files in **/usr/spool/rwho** to generate their status listings.)

rwhod broadcasts and receives status messages using the `rwho` socket as specified in the **/etc/services** file.

When creating these messages, **rwhod** calculates the load average entries for the previous 5-, 10-, and 15-minute intervals. Before broadcasting these messages, **rwhod** converts them to network byte order.

When **rwhod** receives messages on the `rwho` socket, it discards any that do not originate from an `rwho` socket. Additionally, it discards any messages that contain an unprintable ASCII character. When **rwhod** receives a valid message, it places the message in a `whod.`*hostname* file in the **/usr/spool/rwho** directory, overwriting any file with the same name.

Files

/etc/services	Defines Internet socket assignments.
/etc/utmp	Contains status information on users that are logged in to the local host.
/usr/spool/rwho/whod.*hostname*	Contains the latest status information for the host *hostname*.

Related Information

In this book: "**rwho**" on page 2-65 and "**ruptime**" on page 2-63.

The **utmp** file and the **gethostname** routine in *AIX Operating System Technical Reference.*

The **who** command in *AIX Operating System Commands Reference.*

sendmail

Purpose

Provides the server function for sending and receiving remote mail.

Related Information

The **sendmail** command in *AIX Operating System Commands Reference.*

syslogd

Purpose

Provides the server function for reading and logging system messages.

Related Information

The **syslogd** command in *AIX Operating System Commands Reference.*

talkd

Purpose

Provides the server function for the **talk** command.

Syntax

talkd ——|

A5ACC037

Description

The **talkd** command is the server that notifies a user (the ***callee***) that someone else (the ***caller***) wants to initiate a conversation and sets up the conversation if the callee accepts the invitation. The caller initiates the conversation by executing the **talk** command specifying the callee. The callee accepts the invitation by executing the **talk** command specifying the caller. Since the **talkd** server is normally started by the **inetd** server, the **talkd** command should not be issued from the command line.

talkd listens at the socket defined in the **/etc/services** file with an entry beginning with `ntalk`. When **talkd** receives a LOOK_UP request from a local or remote **talk** process, **talkd** scans its internal invitation table for an entry matching the client with a caller.

If such a table entry exists, the client is the callee and **talkd** returns the appropriate rendezvous address to the **talk** process for the callee. The callee process then establishes a stream connection with the caller process.

If no such entry exists, the client is the caller and the client sends an ANNOUNCE request. When **talkd** receives the ANNOUNCE request, **talkd** broadcasts an invitation on the console of the foreign host where callee is logged in unless the caller specifies a particular tty. At approximately 1 minute intervals, **talkd** rebroadcasts the invitation until either the invitation is answered by the callee or the call is canceled by the caller.

Note: **talkd** uses the talk 4.3 protocol, which is not compatible with 4.2 versions of **talk**.

Debugging messages are sent to the **syslogd**. For further information on the files used by this daemon, see **syslogd** in *AIX Operating System Commands Reference*.

Files

/etc/services Defines Internet socket assignments.

/etc/utmp Contains data users currently logged in.

Related Information

In this book: "**talk**" on page 2-70, "**inetd**" on page 3-13, and "**services**" on page 4-66.

The **syslogd** command in *AIX Operating System Commands Reference*.

telnetd

Purpose

Provides the server function for the TELNET protocol.

Syntax

telnetd ──┤

A5ACC001

Description

The **telnetd** command is a server that supports the DARPA standard TELNET virtual terminal protocol. **telnetd** listens at the socket defined in the **/etc/services** file with an entry beginning with `telnet` for requests to connect to the TELNET port. Since the **telnetd** server is normally started by the **inetd** server, the **telnetd** command should not be issued from the command line.

The **telnetd** server assumes that **getty** has opened the server side of a pseudo-terminal (pty) device. The **telnetd** server opens the client side, allowing **getty** to execute a login process.

ptys for use with **telnetd** should be configured using the **devices** command with the following characteristics:

auto enable	true
terminal	terminal type (**$TERM**)
logger	true

When a **telnet** session is started, **telnetd** sends TELNET options to the client (remote) host to indicate an ability to do **remote echo** of characters, to **suppress go ahead**, to receive **terminal type** information, and to **process SAK**.

If the client host agrees, **telnetd** requests its terminal type. On receipt, **telnetd** checks whether the indicated type is supported on the local system. If not, it again requests a terminal type. This terminal type negotiation continues until the remote client sends an acceptable terminal type or until the client sends the same type twice in a row, indicating that it has no other types available.

If the remote client sends the TELNET SAK command, **telnetd** passes the local SAK characters through the pty to invoke the trusted shell.

The **telnetd** server supports the following TELNET options:

- Binary
- Echo/no echo
- Support SAK
- Suppress go ahead
- Timing mark.

telnetd also recognizes the following options for the remote client:

- Binary
- Suppress go ahead
- Terminal type.

File

/etc/ports	Describes configured ports.

Related Information

In this book: "**telnet, tn**" on page 2-72.

The **devices** command in *AIX Operating System Commands Reference.*

The **pty** device driver in *AIX Operating System Technical Reference.*

tftpd

Purpose

Provides the server function for the Trivial File Transfer Protocol.

Syntax

```
tftpd ──┤
```

A5ACC011

Description

The **tftpd** daemon runs the Trivial File Transfer Protocol (TFTP) server. Files sent using TFTP can be found in the directory specified by the full path name given on the **tftp** or **utftp** command line. Since the **tftpd** server is normally started by the **inetd** server, the **tftpd** command should not be issued from the command line.

The use of the **tftp** program requires an account on the the remote system, but **tftp** does not check the account for a password. Due to this lack of authentication information, **tftp** only allows files with read permission for others to be accessed. Files can be written only if they already exist on the remote system and have write permission for others. Since the operation of this command broadens the class of other users to include all users on all hosts that can be reached through the network, use of **tftp** may not be appropriate on all systems.

The **tftpd** server should have a user ID with the fewest privileges possible, which is by convention called **nobody**. To allow use of the **tftpd** server on your host, create a user ID with a separate group that has very restricted privileges. If you decide not to call this user ID **nobody**, edit the **/etc/inetd.conf** file and change the `nobody` on the line that begins `tftp` to the user name you selected. In addition, whether you decide to change the name of the **tftpd** user or not, you must edit the **/etc/inetd.conf** file and remove the comment symbol (#) from the beginning of the `tftp` line. This allows the **tftpd** server to be started when needed by the **inetd** server.

Note: The **tftp** command, **utftp** command, and **tftpd** server are not available in controlled access mode.

Related Information

In this book: "**tftp, utftp**" on page 2-82 and "**inetd**" on page 3-13.

uucpd

Purpose

Provides the server function for using the Basic Network Utilities over an Internet network.

Description

While **uucpd** is not a part of the Interface Program, it is normally started when TCP/IP is initialized. If you need to run this daemon, uncomment the three lines in the **/etc/rc.tcpip** file that follow the phrase `Start up uucpd daemon`. The **uucpd** daemon is then started each time you restart your system.

Related Information

In this book: "**inetd**" on page 3-13.

The **uucpd** command in *AIX Operating System Commands Reference.*

Chapter 4. File Formats

CONTENTS

About This Chapter

This chapter describes the files that are used by Interface Program and that you may need to monitor or modify. The following is a brief description of these files:

/etc/filters	Specifies and authorizes filters for remote printing.
/etc/ftpusers	Specifies local user names that cannot be used by remote FTP clients.
/etc/gated.conf	Contains configuration information for the **gated** daemon.
/etc/gateways	Specifies Internet routing information to **routed**.
/etc/hosts	Defines the IP name and address of the local host and specifies the names and addresses of remote hosts.
/etc/hosts.equiv	Specifies foreign hosts that can execute commands on the local host.
/etc/hosts.lpd	Specifies foreign hosts that can print on the local host.
/etc/inetd.conf	Defines how **inetd** handles Internet service requests.
/etc/named.boot	Defines how **named** sets up the DOMAIN nameserver database.
/etc/named.*	Contains the DOMAIN nameserver database.
/etc/net	Defines all IP characteristics of each adapter used by the Interface Program.
/etc/networks	Specifies the locally known Internet networks.
/etc/protocols	Defines the Internet protocols used on the local host.
/etc/rc.tcpip	Sets up network interfaces, defines the local IP host name and address, initializes Internet routes, and starts daemons that use the Interface Program.
/etc/resolv.conf	Defines DOMAIN nameserver information for local resolver routines.
/etc/services	Defines the sockets and protocols used for Internet services.
$HOME/.netrc	Specifies automatic login information for **ftp** and **rexec**.
$HOME/.rhosts	Specifies remote users that can use a local user account.
$HOME/.3270keys	Defines a user keyboard mapping for TELNET (3270).

In this chapter, the files are organized alphabetically.

filters

Purpose

Specifies and authorizes filters for remote printing.

Synopsis

/etc/filters

Description

The **/etc/filters** file specifies the filters available on a particular host when it acts as a print server. If a client requests a filter that is not listed in **/etc/filters**, the print daemon (**lpd**) sends the file directly to the printer without processing.

The general format of an entry in **/etc/filters** is:

fname fcmd rfflags `|` `print` *rpflags*

The strings in the entry have the following meanings:

fname The name of the filter requested by the **filter** keyword of the **print** command.

fcmd The name of the command to which the filter name (*fname*) maps.

rfflags Required filter flags. These are the flags required by the filter command (*fcmd*).

rpflags Required print flags. These are the flags required by the **print** command as determined by the filter (*fcmd*).

The standard filter name of an AIX system is `pr`. `pr` maps to the **pr** command, which processes the file and passes the required files to the **print** command.

An optional filter name (for a non-AIX system) is `-p`. `-p` maps to the **pr** command, which processes the file and generates the **print** command with `-cp -rm` options for printing the file.

File

/etc/filters Filters for remote printing.

Related Information

In this book: "**lprbe**" on page 2-31 and "**lpd**" on page 3-16.

ftpusers

Purpose

Specifies local user names that cannot be used by remote FTP clients.

Synopsis

/etc/ftpusers

Description

This file contains a list of local user names that the **ftpd** server does *not* allow to be used by remote File Transfer Protocol (FTP) clients. The format of the **/etc/ftpusers** file is a simple list of user names that also appear in the **/etc/passwd** file.

Examples

The following are sample entries in an **/etc/ftpusers** file:

```
root
guest
ftp
joan
uucp
```

File

/etc/ftpusers Disallows remote users.

Related Information

In this book: "**ftpd**" on page 3-5.

gated.conf

Purpose

Contains configuration information for the **gated** daemon.

Synopsis

/etc/gated.conf

Description

The **/etc/gated.conf** file contains configuration information that is read by the **gated** daemon at initialization time. This file contains stanzas that control tracing options, select routing protocols, handle routing information, and determine communications with autonomous systems.

Stanzas can appear in any order in the **gated.conf** file. The following sections illustrate the format of each stanza.

Configuration File Options for Controlling Trace Output

The option that controls trace output is read during **gated** initialization and whenever **gated** receives a **SIGHUP** signal. This option is overriden at initialization time if trace flags are specified on the **gated** command line.

traceflags *flag* [*flag flag* . . .]

The **traceflags** stanza tells the **gated** process what level of trace output is desired. The valid *flags* for tracing are as follows:

internal	Logs all internal errors and interior routing errors.
external	Logs all external errors due to EGP, exterior routing errors, and EGP state changes.
route	Logs all routing changes.
egp	Traces all EGP packets sent and received.
update	Logs all routing updates sent.
rip	Traces all RIP packets received.
hello	Traces all HELLO packets received.

timestamp Prints a timestamp to the log file every 10 minutes.

general Combines the **internal**, **external**, **route**, and **egp** flags.

all Enables all of the listed trace flags.

If more than one **traceflags** stanza is used, the trace flags specified in all stanzas are enabled.

Options for Routing Protocols

This section explains the configuration options for routing protocols, which provide **gated** with instructions on how to manage routing with respect to each protocol.

Note: All references to point-to-point interfaces in the **gated** configuration file must use the *destination* address.

The protocol options are as follows:

RIP *argument*

This stanza tells the **gated** process how to perform the RIP routing protocol. Only one of the following RIP *arguments* is allowed after the **RIP** keyword. Since only the first argument is recognized if more than one is specified, choose the argument that describes the type of RIP routing you need.

A list of the arguments to the **RIP** stanza follows:

yes Performs the RIP protocol, processing all incoming RIP packets and supplying RIP information every 30 seconds only if there are two or more network interfaces.

no Specifies that the RIP protocol not be performed.

supplier Performs the RIP protocol, processing all incoming RIP packets and forcing the supply of RIP information every 30 seconds no matter how many network interfaces are present.

pointopoint Performs the RIP protocol, processing all incoming RIP packets and forcing the supply of RIP information every 30 seconds no matter how many network interfaces are present. When this argument is specified, RIP information is not sent out in a broadcast packet. The RIP information is sent directly to the gateways listed in the **sourceripgateways** stanza, which is described in the following section on options for handling routing information.

quiet Processes all incoming RIP packets, but does not supply any RIP information no matter how many network interfaces are present.

gateway *hopcount* — Processes all incoming RIP packets, supplying RIP information every 30 seconds and announcing the default route (0.0.0.0) with a metric of *hopcount*. The metric should be specified in a value that represents a RIP hop count. With this option set, all other default routes coming from other RIP gateways are ignored. The default route is only announced when actively peering with at least one EGP neighbor and therefore should only be used when EGP is used.

If no **RIP** stanza is specified, RIP routing is not performed.

HELLO *argument*

This stanza configures the HELLO routing protocol for **gated**. The *argument* parallels the RIP arguments, with some minor differences. As with RIP, only one of the following HELLO *arguments* is allowed after the **HELLO** keyword. Since only the first argument is recognized if more than one is specified, choose the argument that describes the type of RIP routing you need.

A list of the arguments to the **HELLO** stanza follows:

yes — Performs the HELLO protocol, processing all incoming HELLO packets and supplying HELLO information every 15 seconds only if there are two or more network interfaces.

no — Specifies that this gateway does not perform the HELLO protocol.

supplier — Performs the HELLO protocol, processing all incoming HELLO packets and forcing supply of HELLO information every 15 seconds no matter how many network interfaces are present.

pointopoint — Performs the HELLO protocol, processing all incoming HELLO packets and forcing supply of HELLO information every 15 seconds no matter how many network interfaces are present. When this argument is specified, HELLO information is not sent out in a broadcast packet. The HELLO information is sent directly to the gateways listed in the **sourcehellogateways** stanza, which is described in the following section on options for handling routing information.

quiet — Processes all incoming HELLO packets, but does not supply any HELLO information regardless of the number of network interfaces present.

gateway *msecs* — Processes all incoming HELLO packets, supplying HELLO information every 15 seconds, and announcing the default route (0.0.0.0) with a time delay of *msecs*. The time delay should be a numeric value, specified in milliseconds. The default route is only announced when actively peering with at least one EGP

neighbor; therefore, this stanza should only be used when running EGP.

If no **HELLO** stanza is specified, HELLO routing is not performed.

EGP yes | no

This stanza allows the processing of EGP by **gated** to be turned on or off.

no Specifies that no EGP processing shoud be performed.

yes Performs all EGP operations.

Note: EGP processing takes place by default, so if no **EGP** stanza is specified, all EGP operations take place.

autonomous system *number*

When performing the EGP protocol, this stanza must be used to specify the autonomous system number. If this number is not specified, **gated** exits immediately with an error message.

egpmaxacquire *number*

When performing the EGP protocol, this stanza specifies the number of EGP peers with whom **gated** performs EGP. The *number* must be greater than 0 and less than or equal to the number of EGP neighbors specified or **gated** exits immediately. If this stanza is omitted, all EGP neighbors are acquired.

egpneighbor *gateway*
metricin *metric*
egpmetricout *egpmetric*
ASin *asin*
ASout *asout*
nogendefault
acceptdefault
defaultout *egpmetric*
validate
intf *interface*
sourcenet *net*

When performing the EGP protocol, this stanza specifies with whom **gated** is to perform EGP. The specified *gateway* can be either a host address in Internet dotted decimal notation, which is less confusing, or a symbolic name from **/etc/hosts**. Each EGP neighbor should have its own **egpneighbor** stanza and is acquired in the order listed in the **gated.conf** file. Possible arguments to this stanza are:

metricin *delay* Specifies the internal time delay to be used as a metric for all of the routes learned from this neighbor. The *delay* should be specified as a time delay from 0 to 30000. If this keyword and the

validate keyword are not used, the internal metric used is the EGP distance multiplied by 100.

egpmetricout *egpmetric*	Sets the EGP distance used for all nets advertised to this neighbor. The *egpmetric* should be specified as an EGP distance in the range of 0 to 255. If this keyword is not specified, the internal time delay for each route is converted to an EGP distance by division by 100, with distances greater than 255 being set to 255.
ASin *autonsys*	Verifies the autonomous system number of this neighbor. If the *autonsys* number specified in neighbor acquisition packets does not verify, an error message is generated refusing the connection. If this keyword is not specified, no verification of autonomous system numbers is done.
ASout *autonsys*	Specifies the autonomous system number in EGP packets sent to this neighbor. If an **ASout** stanza is not specified, the *autonsys* number specified in the **autonomoussystem** stanza is used. This stanza is reserved for a special situation that occurs between the ARPANET and NSFnet, and is not normally used.
nogendefault	Specifies that this neighbor should not be considered for the internal generation of default when **RIP gateway** or **HELLO gateway** is used. If not specified, the internal default is generated when actively peering with this neighbor.
acceptdefault	Indicates that the default route (net 0.0.0.0) should be considered valid when received from this neighbor. If this keyword is not specified, on reception of the default route, **gated** displays a warning message and ignores the route.
defaultout *egpmetric*	Specifies that the internally generated default may be passed to this EGP neighbor at the specified distance. The distance should be specified as an EGP distance from 0 to 255. A default route learned from another gateway is not propagated to an EGP neighbor. Without this keyword, no default route is passed via EGP. The **acceptdefault** keyword should not be specified when the **defaultout** keyword is used. The EGP metric specified in the **egpmetricout** keyword does not apply when the **defaultout** keyword is used. The default route always uses the metric specified by the **defaultout** keyword.

validate Specifies that all networks received from this EGP neighbor must be defined in a **validAS** stanza that also specifies the autonomous system of this neighbor. Networks without a **validAS** stanza are ignored after a warning message is printed.

intf *interface* Defines the interface used to send EGP packets to this neighbor. This keyword is only used when there is no common net or subnet with this EGP neighbor. This keyword is present for testing purposes and does not imply correct operation when peering with an EGP neighbor that does not share a common net or subnet.

sourcenet *net* Specifies the source network to be used in EGP poll packets sent to this neighbor. If this keyword is not specified, the network (not subnet) of the interface used to communicate with this neighbor is used. This keyword is present for testing purposes and does not imply correct operation when used.

Options for Handling Routing Information

The following configuration file keywords determine how **gated** handles both incoming and outgoing routing information.

trustedripgateways *gateway* [*gateway gateway* . . .]
trustedhellogateways *gateway* [*gateway gateway* . . .]

When these stanzas are specified, **gated** only listens to RIP or HELLO information, respectively, from these RIP or HELLO gateways. *gateway* can be either an Internet address in dotted decimal notation, which avoids confusion, or a symbolic name from **/etc/hosts**. Note that the propagation of routing information is not restricted by this stanza.

sourceripgateways *gateway* [*gateway gateway* . . .]
sourcehellogateways *gateway* [*gateway gateway* . . .]

With these stanzas, **gated** sends RIP or HELLO information directly to the gateways specified. If **pointopoint** is specified in the **RIP** or **HELLO** stanzas defined earlier, **gated** only sends RIP or HELLO information to the specified gateways and does *not* send out any information using the broadcast address. If **pointopoint** is not specified in those stanzas and **gated** is supplying RIP or HELLO information, **gated** sends information to the specified gateways, as well as broadcasting it using a broadcast address.

noripoutinterface *intfaddr* [*intfaddr intfaddr* . . .]
nohellooutinterface *intfaddr* [*intfaddr intfaddr* . . .]
noripfrominterface *intfaddr* [*intfaddr intfaddr* . . .]
nohellofrominterface *intfaddr* [*intfaddr intfaddr* . . .]

These stanzas turn routing protocols on and off on a per interface basis. A **noripfrominterface** or **nohellofrominterface** means that no RIP or HELLO information is accepted coming into the listed interfaces from another gateway. A **noripoutinterface** or **nohellooutinterface** means that no RIP or HELLO knowledge is sent out of the listed interfaces. The *intfaddr* should be an Internet address in dotted decimal notation.

passiveinterfaces *intfaddr* [*intfaddr intfaddr* . . .]

This stanza stops **gated** from timing out the interfaces whose address are listed in Internet dotted decimal notation by the *intfaddr* arguments. These interfaces are always considered up and working.

This stanza is used because **gated** times out an interface when no RIP, HELLO, or EGP packets are being received on that particular interface, in order to dynamically determine if an interface is functioning properly. PSN interfaces send a RIP or HELLO packet to themselves to determine if the interface is properly functioning, since the delay between EGP packets may be longer than the interface timeout. Interfaces that have timed out automatically have their routes re-installed when routing information is again received over the interface.

If **gated** is not a RIP or HELLO supplier, all interfaces are not aged and the **passiveinterfaces** stanza automatically applies to all interfaces.

interfacemetric *intfaddr metric*

This stanza allows the specification of an interface metric for the listed interface. On systems that support interface metrics, this stanza overrides the kernel's metric. On systems that do not support an interface metric, this feature allows the specification of one. The interface metric is added to the true metric of each route that comes in via routing information from the listed interface. The interface metric is also added to the true metric of any information sent out via the listed interface. The metric of directly attached interfaces is also set to the interface metric, and routing information broadcast about directly attached nets is based on the interface metric specified. This stanza is required for each interface on which an interface metric is desired.

reconstmetric *intfaddr metric*

This stanza provides hooks for fallback routing in **gated**. If this stanza is used, the metrics of the routes contained in any RIP information coming into the listed interface are set to the specified *metric*. Metric reconstitution should not be used lightly, since it could be a major contributor in the formation of routing loops. Any route that has a metric of infinity is not reconstituted and left as infinity.

Note: The **reconstmetric** stanza should be used with extreme caution.

fixedmetric *intfaddr proto* **rip** | **hello** *metric*

The **fixedmetric** stanza also provides hooks for fallback routing in **gated**. If this stanza is used, all routing information sent out the specified interface has a metric of *metric*. For RIP, specify the metric as a RIP hop count from 0 to infinity. For HELLO, specify the metric as a HELLO delay in milliseconds from 0 to infinity. Any route that has a metric of infinity is left as infinity.

Note: Fixed metrics should be used with extreme caution.

donotlisten *net* **intf** *addr* [*addr* . . .] **proto rip** | **hello**
donotlistenhost *host* **intf** *addr* [*addr* . . .] **proto rip** | **hello**

The **donotlisten** stanza contains the following information: keyword **donotlisten**, followed by a network number *net*, which should be in dotted decimal notation, followed by the keyword **intf**. Next is a list of interfaces in dotted decimal notation, then the keyword **proto**, followed by **rip** or **hello**.

This stanza means that any information regarding *net* that comes in via the specified protocols *and* from the specified interfaces is ignored. The keyword **all** can be used after the **intf** keyword to specify all interfaces on the system. For example:

```
donotlisten 10.0.0.0 intf 128.84.253.200 proto rip
```

means that any RIP information about network 10.0.0.0 coming in via interface `128.84.253.200` will be ignored. One stanza is required for each network on which this restriction is desired. In addition:

```
donotlisten 26.0.0.0 intf all proto rip hello
```

means that any RIP and HELLO information about net `26.0.0.0` coming in via any interface will be ignored.

The **donotlistenhost** stanza is defined in the same way, except that a host address is provided instead of a network address. Restrictions on routing updates are applied to the specified host route learned of by the specified routing protocol or protocols.

listen *net* **gateway** *addr* [*addr* . . .] **proto rip** | **hello**
listenhost *host* **gateway** *addr* [*addr* . . .] **proto rip** | **hello**

The **listen** and **listenhost** stanzas mean to only listen to information about a network or host on the specified protocol or protocols *and* from the listed gateways.

These stanzas read as follows: keyword **listen** or **listenhost**, followed by a network or host address, respectively, in dotted decimal notation. Next is the **gateway** keyword, and a list of gateways in dotted decimal notation, then the keyword **proto**, followed by **rip** or **hello**. For example:

```
listen 128.84.0.0 gateway 128.84.253.3 proto hello
```

indicates that any HELLO information about network 128.84 that comes in via gateway 128.84.253.3 is accepted. Any other information about network 128.84 from any other gateway is rejected. One stanza is needed for each net to be restricted.

Also, the stanza:

```
listenhost 26.0.0.15 gateway 128.84.253.3 proto rip
```

means that any information about host 26.0.0.15 must come via RIP from gateway 128.84.253.3. All other information regarding this host is ignored.

announce *net intf addr* [*addr* . . .] *proto type* [*egpmetric*]
announcehost *host intf addr proto type* [*egpmetric*]
noannounce *net intf addr* [*addr* . . .] *proto type* [*egpmetric*]
noannouncehost *host intf addr proto type* [*egpmetric*]

These stanzas allow restriction of the networks and hosts that are announced and the protocols that announce them. The **announce**{**host**} and **noannounce**{**host**} stanzas can not be used together on the same interface. With the **announce**{**host**} stanza, **gated** only announces the nets or hosts that have an associated **announce** or **announcehost** stanza with the appropriate protocol. With the **noannounce{host}** stanza, **gated** announces everything *except* those nets or hosts that have an associated **noannounce** or **noannouncehost** stanza. These stanzas provide a choice of announcing only what is on the announce list or everything except those nets on the noannounce list on a per interface basis.

The arguments are the same as in the **donotlisten** stanza except **egp** may be specified in the *proto* field. The *type* can be **rip**, **hello**, **egp**, or any combination of the three. When **egp** is specified in the *proto* field, an EGP metric must be specified. This is the metric at which **gated** announces the listed net via EGP.

Note that these are not static route entries. These restrictions only apply if the net or host is learned via one of the routing protocols. If a restricted network suddenly becomes unreachable and goes away, announcement of this net stops until it is learned again.

Only one **announce**{**host**} or **noannounce**{**host**} stanza may be specified per network or host. A network or host cannot, for instance, be announced via HELLO out one interface and via RIP out another.

Some example **announce** stanzas might include:

```
announce 128.84 intf all proto rip hello egp egpmetric 0
announce 10.0.0.0 intf all proto rip
announce 0.0.0.0 intf 128.84.253.200 proto rip
announce 35.0.0.0 intf all proto rip egp egpmetric 3
```

With only these four **announce** stanzas in the configuration file, the **gated** process only announces these four nets. Network `128.84.0.0` is announced via RIP and HELLO to all interfaces and via EGP with a metric of 0. Network `10.0.0.0` is announced via RIP to all interfaces. Network `0.0.0.0` (default) is announced by RIP out interface `128.84.253.200` only. Network `35.0.0.0` is announced via RIP to all interfaces and announced via EGP with a metric of 3. These are the only nets that are broadcast by this gateway. Once the first **announce** stanza is specified, only the networks with **announce** stanzas are broadcast, including local subnets. Once an **announce**{**host**} or **noannounce**{**host**} stanza has an **all** specified after an **intf** keyword, that stanza is applied globally and the option of having per interface restrictions is lost. If no routing announcement restrictions are desired, **announce** stanzas should not be used. All information learned is then propagated out. Note that announcement has no affect on the information to which **gated** listens. Any network that does not have an **announce** stanza is still added to the kernel routing tables, but it is not announced via any of the routing protocols. To stop networks from being added to the kernel, the **donotlisten** stanza may be used.

As another example:

```
announce 128.84 intf 128.59.2.1 proto rip
noannounce 128.84 intf 128.59.1.1 proto rip
```

indicates that on interface `128.59.2.1`, only information about network `128.84.0.0` is announced via RIP, but on interface `128.59.1.1`, all information is announced, except `128.84.0.0` via RIP.

The stanzas:

```
noannounce 128.84 intf all proto rip hello egp egpmetric 0
noannounce 10.0.0.0 intf all proto hello
```

mean that except for the two specified nets, all networks are propagated. Specifically, network `128.84.0.0` is not announced on any interface via any protocols. Knowledge of network `128.84.0.0` is not sent anywhere. Network `10.0.0.0` is not announced via HELLO to any interface. The second stanza also implies that network `10.0.0.0` is announced to every interface via RIP. This net is also broadcast via EGP with the metric specified in the **defaultegpmetric** stanza.

defaultegpmetric *number*

This stanza defines a default EGP metric to use when there are no routing restrictions. Without routing restrictions, **gated** announces all networks learned via HELLO or RIP via EGP with this specified default EGP metric. If this stanza is not used, the default EGP metric is set to 255, which causes any EGP advertised route of this nature to be ignored. When there are no routing restrictions, any

network with a direct interface is announced via EGP with a metric of 0. Note that this does not include subnets, but only the non-subnetted network.

defaultgateway *gateway* [*metric*] *proto* [**active** | **passive**]

This stanza defines a default gateway, which is installed in the kernel routing tables during initialization and reinstalled whenever information about the default route is lost. This route is installed with the time delay equivalent of a RIP metric of 15, unless another metric is specified with the *metric* argument.

If **RIP gateway** or **HELLO gateway** is in use, this default route is deleted.

An **active** default route is overridden by any other default route learned via another routing protocol. A **passive** default route is only overridden by a default route with a lower metric. In addition, an **active** default route is not propagated in routing updates, while a **passive** default route is propagated.

The specified *gateway* should be an address in Internet dotted decimal notation. The *metric* is optional and should be a time delay from 0 to 30000. If a *metric* is not specified, the equivalent time delay to a RIP metric of 15 is used. The *proto* argument should be one of **rip**, **egp**, or **hello**. The *proto* field initializes the protocol by which the route was learned. In this case the *proto* field is unused but remains for consistency.

net *netaddr* **gateway** *addr* **metric** *hopcnt* **rip** | **egp** | **hello**
host *hostaddr* **gateway** *addr* **metric** *hopcnt* **rip** | **egp** | **hello**

The **net** and **host** stanzas install a static route to net *netaddr* or host *hostaddr* through gateway *addr* at a metric of *hopcnt* learned via RIP, HELLO, or EGP. Again, dotted decimal notation should be used for the addresses. These routes are installed in the kernel's routing table and are never affected by any other gateway's RIP or HELLO announcements. The protocol by which they were learned is important if the route is to be announced via EGP. If the protocol is RIP or HELLO and there are no routing restrictions, then this route is announced by EGP with a metric of **defaultegpmetric**. If the protocol keyword is **egp** and there are no routing restrictions, then this route is announced by EGP with a metric of *hopcnt*.

egpnetsreachable *net* [*net net . . .*]

This stanza provides a *soft restriction* to **gated**. It cannot be used when the **announce** or **noannounce** stanzas are used. With no restrictions, **gated** announces all routes learned from RIP and HELLO via EGP. The **egpnetsreachable** stanza restricts EGP announcement to those networks listed in the stanza. The metric used for the HELLO and RIP learned routes is the value given in the **defaultegpmetric** stanza. If this stanza does not specify a value, the value is set to 255. With the **egpnetsreachable** stanza, unique EGP metrics cannot be set for each network. The **defaultegpmetric** is used for all networks except those that are directly connected, which use a metric of 0.

martiannets *net* [*net net* . . .]

This stanza appends to **gated**'s list of ***martian*** networks, which are those that are known to be invalid and should be ignored. When **gated** receives information about one of these networks through any means, it immediately ignores it. If **external** tracing is enabled, a message is printed to the trace log. Multiple occurences of the **martiannets** stanza accumulate.

The initial list of martian networks provided by **gated** contains the following networks: 127.0.0.0, 128.0.0.0, 191.253.0.0, 192.0.0.0, 223.255.255.0, and 224.0.0.0.

Options for Autonomous System Routing

In the internal routing tables, **gated** maintains the autonomous system number from which each route was learned. Autonomous systems are used only when an exterior routing protocol is in use, in this case EGP. Routes are tagged with the autonomous system number of the EGP peer from which they were learned. Routes learned via the interior routing protocols, RIP and HELLO, are tagged with the autonomous system number specified in the **autonomoussystem** stanza of the **gated.conf** file.

Note: The **gated** server does not normally propagate routes learned from exterior routing protocols to interior routing protocols, since some gateways do not have adequate validation of routing information they receive. Some of the following stanzas allow exterior routes to be propagated via interior protocols. Therefore, it is imperative that utmost care be taken when allowing the propagation of exterior routes.

The following stanzas provide limited control over routing based on autonomous system number.

validAS *net* **AS** *system* **metric** *number*

The **validAS** stanza is used for validation of networks from a certain autonomous system. When an EGP update is received from a neighbor that has the **validate** keyword specified in the associated **egpneighbor** stanza, a **validAS** stanza is searched for that defines that network and the autonomous system number of the EGP neighbor. If the appropriate **validAS** stanza is located, the network is considered for addition to the routing table with the specified metric. If a **validAS** stanza is not located, a warning message is printed and the network is ignored.

A network may be specified in several **validAS** stanzas as being associated with several different autonomous systems.

announcetoAS *autonsys1* **ASlist** *autonsys2* [*autonsys3* . . .]
noannouncetoAS *autonsys1* **ASlist** *autonsys2* [*autonsys3* . . .]

The **announcetoAS** and **noannouncetoAS** stanzas control the exchange of routing information between different autonomous systems. Normally **gated** does not propagate routing information between autonomous systems. The exception to this is that routes learned from **gated**'s own autonomous system via RIP and HELLO are propagated via EGP. These stanzas allow information learned via EGP

from one autonomous system to be propagated via EGP to another autonomous system or via RIP and HELLO to **gated**'s own autonomous system.

If the **announcetoAS** stanza is specified, information learned via EGP from autonomous systems *as1*, *as2*, *as3*, and so on is propagated to autonomous system *as0*. If **gated**'s own autonomous system, as specified in the **autonomoussystem** stanza, is specified as *as0*, this information is propagated via RIP and HELLO. Routing information from autonomous systems not specified in the **ASlist** are not propagated to autonomous system *as0*.

If the **noannouncetoAS** stanza is specified, information learned via EGP from all autonomous systems except *as1*, *as2*, *as3*, and so on is propagated to autonomous system *as0*. If **gated**'s own autonomous system is specified as *as0*, this information is not propagated via RIP and HELLO.

Only one **announcetoAS** or **noannounceAS** stanza may be specified per target autonomous system.

Example

A sample **gated.conf** file for a **gated** server that performs only EGP routing might contain the following entries:

```
RIP     no
HELLO   no
EGP     yes
#
traceflags      internal external route egp update
autonomoussystem        178
egpneighbor             192.100.9.1
defaultegpmetric        132
#
net 192.200.9 gateway 192.101.9.3 metric 50 rip
egpnetsreachable 192.200.9 192.101.9

# Static routes

host 129.140.46.1      gateway 192.100.9.1    metric 5 rip
host 192.102.9.2       gateway 192.100.9.1    metric 5 rip
host 192.104.9.2       gateway 192.100.9.1    metric 5 rip
host 149.140.3.12      gateway 192.100.9.1    metric 5 rip
host 129.140.3.12      gateway 192.100.9.1    metric 5 rip
host 129.140.3.13      gateway 192.100.9.1    metric 5 rip
```

```
host 129.140.3.14       gateway 192.100.9.1    metric 5 rip
host 192.3.3.54         gateway 192.101.9.3    metric 5 rip
```

File

/etc/gated.conf Contains configuration information for the **gated** daemon.

Related Information

In this book: "**gated**" on page 3-9.

gateways

Purpose

Specifies Internet routing information to **routed**.

Synopsis

/etc/gateways

Description

The **/etc/gateways** file identifies gateways for the **routed** command. Ordinarily, **routed** queries the network, building routing tables from routing information transmitted by other hosts that are directly connected to the network. However, there may be gateways that **routed** cannot identify through its queries (***distant*** gateways). Such gateways should be identified in **/etc/gateways**, which **routed** reads when it starts.

The general format of an entry in **/etc/gateways** is:

destination name1 `gateway` *name2* `metric` *value type*

Following is a brief description of each element in an **/etc/gateways** file entry:

destination A keyword that indicates whether the route is to a network or to a specific host. The two possible keywords are **net** and **host**.

name1 The name associated with *destination*. *name1* can be either a symbolic name (as used in **/etc/hosts** or **/etc/networks**) or an Internet address specified in dotted decimal format.

gateway Indicator that the following string identifies the gateway host.

name2 Name or address of the gateway host to which messages should be forwarded.

metric Indicator that the next string represents the ***hop count*** to the destination host or network.

value The hop count, or number of gateways from the local network to the destination network.

type A keyword that indicates whether the gateway should be treated as active, passive, or external. The three possible keywords are:

active An active gateway is treated like a network interface. That is, it is expected to exchange RIP routing information. Information about it is maintained in the internal routing tables as long as it is active and is included in any routing information that is transmitted via RIP. If it does not respond for a period of time, the route associated with it is deleted from the internal routing tables.

passive A passive gateway is not expected to exchange RIP routing information. Information about it is maintained in the routing tables indefinitely and is included in any routing information that is transmitted via RIP.

external An external gateway is identified to inform **routed** that another routing process will install such a route and that alternative routes to that destination should not be installed. Information about external gateways is not maintained in the internal routing tables and is not transmitted via RIP.

Note: These routes must be to networks.

Examples

Following are five sample **/etc/gateways** entries:

1. A route to a network, `net2`, through the gateway `host4`. The hop count metric to `net2` is `4` and the gateway is treated as `passive`.

   ```
   net net2 gateway host4 metric 4 passive
   ```

2. A route like the one in the previous example except that it is to a specific host (rather than to a network):

   ```
   host host2 gateway host4 metric 4 passive
   ```

3. A route to a specific host, `host10`, through the gateway `192.100.11.5`. The hop count metric to `host10` is 9 and the gateway is treated as `active`.

   ```
   host host10 gateway 192.100.11.5 metric 9 active
   ```

4. A route like the one in the previous example except that the gateway is treated as `passive` (rather than `active`):

   ```
   host host10 gateway 192.100.11.5 metric 9 passive
   ```

5. A route to a network, `net5`, through the gateway `host7`. The hop count metric to `net5` is `11` and the gateway is treated as `external` (that is, it is not advertised via RIP but is advertised via an unspecified routing protocol).

```
net net5 gateway host7 metric 11 external
```

Files

/etc/gateways	Internet routing information for **routed**.
/usr/lpp/tcpip/samples/gateways	Sample **gateways** file.

Related Information

In this book: "**routed**" on page 3-28.

hosts

Purpose

Defines the IP name and address of the local host and specifies the names and addresses of remote hosts.

Synopsis

/etc/hosts

Description

This file contains the IP host names and addresses for the local host and other hosts in the Internet network. This file is used to resolve a name into an address (that is, to translate a *hostname* into its Internet address). How the data in this file is used differs, depending on whether the local host is using a DOMAIN name server or no name server at all.

When the local host is using the DOMAIN protocol, the resolver routines query a remote DOMAIN name server before searching this file. In a flat network with no name server, the resolver routines search this file for host name and address data.

When no name server is used, each entry in the file is usually of the form:

address hostname hostname hostname hostname

where *address* is an IP address specified in either dotted decimal or octal format, and *hostname* is a string with a maximum length of 24 characters and no embedded blanks. (See "Host Names" on page 1-28.) Multiple *hostnames* (or aliases) can be specified as long as the total number of characters does not exceed 100 characters; each entry must be contained on one line. Additionally, this file can contain two additional entries that define reserved (or ***well-known***) host names. These host names are:

timeserver Identifies a remote timeserver host (see "**setclock**" on page 2-68).

printserver Identifies the default host for receiving print requests (see "**lprbe**" on page 2-31).

When using the DOMAIN protocol, each entry in the **hosts** file is usually of the form:

address hostname hostname hostname hostname

where *address* is an IP address specified in either dotted decimal or octal format, and each *hostname* is a host name specified in either relative or absolute domain name format. (For more information on the formats of domain names, see "Domain Name Protocol

(DOMAIN)" on page 1-18.) If you specify the absolute domain name, the portion of the name preceding the first dot (.) has a maximum length of 24 characters and cannot contain blanks. For both formats of the name, the total number of characters cannot exceed 100 characters, and each entry must be contained on one line.

Note: The local **/etc/resolv.conf** file defines where DOMAIN name servers are, and the nameserver database defines where Internet services are available. Therefore, there is no need to define well-known hosts in the **hosts** file when using the DOMAIN protocol.

Examples

The following examples show **/etc/hosts** files for two different types of networks:

- A network that has no name server in operation.
- A network that has a Domain Name Protocol name server running.

In these examples, the name of the local host is the first line in each **/etc/hosts** file. This is to help you identify the host whose file is being displayed; your host does not have to be defined on the first line of your **/etc/hosts** file.

1. The following sample entries might be contained in the **/etc/hosts** files for two different hosts on a network that is not running a DOMAIN name server:

Host 1

```
185.300.10.1  host1
185.300.10.2  host2
185.300.10.3  host3
185.300.10.4  host4 merlin
185.300.10.5  host5 arthur king
```

Host 2

```
185.300.10.2  host2
185.300.10.1  host1
185.300.10.3  host3
185.300.10.4  host4 merlin
185.300.10.5  host5 arthur king
```

In this sample network with no name server, the **/etc/hosts** file for each host must contain the Internet address and host name for each host on the network. Any host that is not listed cannot be accessed. The host at Internet address `185.300.10.4` in this example can be accessed by either name `host4` or `merlin`, and the host at

Internet address 185.300.10.5 can be accessed by any of the names host5, arthur, or king.

2. Following are sample entries in the **/etc/hosts** files for two different hosts in a DOMAIN network:

Host 1		
192.100.10.1	host1.abc.aus.ibm.com	host1
192.100.10.2	host2	
192.100.10.3	host3	
192.100.10.4	host4	
192.100.10.5	host5	
128.114.1.15	name1.xyz.aus.ibm.com	
128.114.1.14	name2.xyz.aus.ibm.com	
128.114.1.16	name3.xyz.aus.ibm.com	

Host 2	
192.100.10.2	host2.abc.aus.ibm.com
192.100.10.1	host1

In this sample DOMAIN network, host1 is the name server. The **/etc/hosts** file for host1 contains address entries for all hosts in the network, and the **named.*** files contain the DOMAIN database. As the name server, host1 runs the **named** daemon. The entries in the host1 **/etc/hosts** file that begin with 128.114 indicate that host1 also resolves names for hosts on more than one network. For hosts that reside in different domains, the full domain name must be specified in the **/etc/hosts** file of the remote system.

The **/etc/hosts** file of host2 contains an address entry for host2 itself that includes the full domain name of the host. The file for host2 also contains an entry for the name server for this network, host1. host3 through host5 on this network have similar **/etc/hosts** file to the one for host2, except that the first entry contains the full domain name of each of those hosts.

File

/etc/hosts Contains host names and addresses.

Related Information

In this book: "Names and Addresses" on page 1-27.

The **gethostbyaddr,...** routines in *AIX Operating System Technical Reference.*

hosts.equiv

Purpose

Specifies foreign hosts that can execute commands on the local host.

Synopsis

/etc/hosts.equiv

Description

The **/etc/hosts.equiv** file defines which foreign hosts are permitted to execute certain commands on the local host without supplying a password when the local and remote user names are the same. In a sense, users on the foreign hosts become *equivalent* to users on the local host with the same user name. The format of the **/etc/hosts.equiv** file is a simple list of host names.

Only the **lpd**, **rlogind**, and **rshd** commands check for the existence of a **hosts.equiv** file.

If you are using a DOMAIN name server and you want to grant access to hosts in a different domain, you must specify the full domain name of each host.

Example

Following are sample entries in an **/etc/hosts.equiv** file:

```
host1
host2
host3
host4
```

File

/etc/hosts.equiv Contains list of equivalent hosts.

Related Information

In this book: "**lpd**" on page 3-16, "**rlogind**" on page 3-26, and "**rshd**" on page 3-33.

hosts.lpd

Purpose

Specifies foreign hosts that can print on the local host.

Synopsis

/etc/hosts.lpd

Description

The **/etc/hosts.lpd** file defines which foreign hosts are permitted to print on the local host. The foreign hosts listed in this file are not given the full privileges given to files listed in the **/etc/hosts.equiv** file. The format of the **/etc/hosts.lpd** file is a simple list of host names.

Example

Following are sample entries in an **/etc/hosts.lpd** file:

```
host12
host23
host25
```

File

/etc/hosts.lpd Specifies foreign hosts that can print on the local host.

Related Information

In this book: "**lpd**" on page 3-16.

inetd.conf

Purpose

Defines how **inetd** handles Internet service requests.

Synopsis

/etc/inetd.conf

Description

This file is the default configuration file for the **inetd** server. If **inetd** is started without specifying an alternate configuration file, **inetd** reads this file for information on how to handle Internet service requests. **inetd** reads its configuration file only when **inetd** starts or when **inetd** receives a **SIGHUP** signal. Each line in the **inetd** configuration file defines how to handle one Internet service request. Each line is of the form:

servname socktype protname wait/nowait username serverpath serverargs

where the fields are separated by spaces or tabs and have the following meanings:

servname
: is the name of an Internet service defined in the **/etc/services** file. For services provided internally by **inetd**, this name must be the official name of the service. That is, the name must be identical to the first entry on the line that describes the service in the **/etc/services** file.

socktype
: is the type of socket used for the service and can be either `stream` for a stream socket or `dgram` for a datagram socket.

protname
: is the name of an Internet protocol defined in the **/etc/protocols** file. For example, use `tcp` for a service that uses the TCP protocol and `udp` for a service that uses the UDP protocol.

wait/nowait
: is either `wait` or `nowait` for datagram sockets and `nowait` for stream sockets. This field determines whether or not **inetd** waits for a datagram server to release the socket before continuing to listen at the socket.

username
: is the user name that **inetd** should use to start the server. This allows a server to be given less permission than `root`.

serverpath
: is the full pathname of the server that **inetd** should execute to provide the service. For services that **inetd** provides internally, this field should be `internal`.

serverargs are the command line arguments that **inetd** should use to execute the server. These arguments begin with the name of the server used. For services that **inetd** provides internally, this field should be empty.

Examples

The following are sample entries in the **/etc/inetd.conf** file for an **inetd** server that uses **ftpd** for servicing `ftp` requests, uses **talkd** for `ntalk` requests, and provides `time` requests internally. (For more information on the functions that **inetd** provides internally, see "**inetd**" on page 3-13.)

```
ftp    stream   tcp   nowait   root   /usr/lpp/tcpip/etc/ftpd     ftpd
ntalk  dgram    udp   wait     root   /usr/lpp/tcpip/etc/talkd    talkd
time   stream   tcp   nowait   root   internal
time   dgram    udp   wait     root   internal
```

File

/etc/inetd.conf Contains configuration information for **inetd** daemon.

Related Information

In this book: "**inetd**" on page 3-13.

named.boot

Purpose

Defines how **named** initializes the DOMAIN nameserver database.

Synopsis

/etc/named.boot

Description

This file is the default configuration (or boot) file for the **named** server. If **named** is started without specifying an alternate file, **named** reads this file for information on how to set up the local name server database. This database is directly accessed by local kernel resolver routines on the name server host and is indirectly accessed through **named** service requests by remote hosts.

Note: **named** reads the boot file only when **named** starts or when **named** receives a **SIGHUP** signal.

The records in the **named** boot file tell **named** what type of server it is, which domains (or zones of authority) it has authority over, and where to get the data for initially setting up its database.

The name server first needs to know the root name server, which is the authority server for the network. The root name server is established in the **named.boot** file by specifying the root server filename (**named.ca**) as the cache for this name server.

The general format of each line in this file is:

type domain source

where the *type* field determines the type of information and can have one of the following values:

domain Indicates that the following *domain* entry is the name of the default domain. When **named** receives a query with a domain or host name that does not end with period (**.**), **named** appends the default domain name to the queried name.

primary Indicates that the local **named** server is the primary name server for the domain specified in the *domain* field and that **named** is to get the data describing the domain from the file specified in the *source* field.

secondary Indicates that the local **named** server is a secondary name server for the domain specified in the *domain* field and that **named** is to get the data describing the domain from one or more remote primary name servers using the Internet address or addresses specified in the *source* field. **named** tries each address in the order listed until it successfully receives the data from one of the name servers.

cache Indicates that the local **named** server is a caching name server for the domain specified in the *domain* field and that **named** is to get the data describing the domain from the file specified in the *source* field.

Note: **named** does not provide other hosts with the information contained in a cache file. Cache files are usually used for listing the name servers for domains higher than the local domain.

Comments in the boot file begin with a semicolon (;) and end at the end of the line.

Note: Any data files referenced in the **named** boot file must be in Standard Resource Record Format. These data files can have any name. However, for convenience in maintaining the **named** database, they are generally given names in the following form: **/etc/named.***. For this reason, the general format of **named** data files is described in "**named.***" on page 4-37.

Examples

In the following examples, the names and IP addresses of each of the servers are:

```
Primary name server    :  venus   (192.9.201.1)
Secondary name server  :  kronos  (192.9.201.2)
```

1. The **/etc/named.boot** file for **venus**, the primary name server, contains these entries:

```
; Boot file for primary name server
;
; type        domain                       source file or host
domain        abc.aus.ibm.com
cache         .                              /etc/named.ca
primary       abc.aus.ibm.com                /etc/named.abcdata
primary       xyz.aus.ibm.com                /etc/named.xyzdata
primary       201.9.192.in-addr.arpa         /etc/named.abcrev
primary       100.114.128.in-addr.arpa       /etc/named.xyzrev
primary       0.0.127.in-addr.arpa           /etc/named.local
```

2. The **/etc/named.boot** file for **kronos**, the secondary name server, contains these entries:

```
; Boot file for secondary name server
;
; type         domain                    source file or host
domain         abc.aus.ibm.com
cache          .                          /etc/named.ca
secondary      abc.aus.ibm.com            192.9.201.1 192.9.201.2
secondary      xyz.aus.ibm.com            192.9.201.1 192.9.201.2
secondary      201.9.192.in-addr.arpa     192.9.201.1 192.9.201.2
secondary      100.114.128.in-addr.arpa 192.9.201.1 192.9.201.2
primary        0.0.127.in-addr.arpa       /etc/named.local
```

3. The **/etc/named.boot** file for **hera**, a cache-only server, contains these entries:

```
; Boot file for cache only server
;
; type         domain                    source file or host
domain         abc.aus.ibm.com
cache          .                         /etc/named.ca
primary        0.0.127.in-addr.arpa      /etc/named.local
```

Files

/etc/named.boot	Contains configuration information for **named** server.
/usr/lpp/tcpip/samples/named.boot	Sample **named.boot** file.

Related Information

In this book: "**named**" on page 3-20 and "**named.***" on page 4-37.

named.*

Purpose

Define the data that **named** uses to initialize the DOMAIN nameserver database.

Synopsis

/etc/named.*

Description

Any data files referenced in the **named.boot** file must be in Standard Resource Record Format. While these data files can have any names, they are generally given names of the form **/etc/named.***. This naming convention is used for convenience in maintaining the **named** database.

Two **awk** scripts, **addrs.awk** and **hosts.awk**, are provided in the **/usr/lpp/tcpip/samples** directory to assist you in converting your existing **/etc/hosts** database to a DOMAIN database. Refer to these files for more information on the conversion.

The records in the **named** data files are called resource records. Except for comments (starting with a semicolon (;) and ending at the end of the line), the resource records in the data files generally follow the format of the resource records that **named** returns in response to queries from resolver routines.

The general format of each resource record in a **named** data file is:

name time-to-live address-class record-type record-data

where:

name is the name of the domain record (that is, either a host or domain name). This field starts in the first column and can be blank for some records. If the field is blank, the *name* field from the previous record applies to the current record. The following characters have special meanings in the *name* field:

- . A free-standing period refers to the current domain.
- @ A free-standing at sign (@) refers to the current origin.
- .. Two free-standing periods refer to the null domain name of the root domain.

* An asterisk in a domain name indicates wildcarding. For example, a domain name of the form `*.domain` refers to all domains below *domain*.

time-to-live is an optional field that defines how long (in seconds) the data in the resource record is to be kept in the database. If this field is not defined, the default value is the **minimum** field as defined in the ***Start of Authority*** record for this zone.

address-class is the address class of the resource record and can be either `IN` for Internet addresses or `ANY` for all address classes.

record-type defines the type of data contained in the *record-data* field.

record-data is one or more fields that contain the data for the resource record.

In addition to the characters with special meanings in the *name* field, the following characters also have special meanings:

; A semicolon begins a comment, which continues to the end of the line.

X A backslash before any character other than a digit (0-9) quotes the following character so that the character's special meaning is ignored.

DDD A backslash before a three-digit decimal number is assumed to be an eight-bit character and is not checked for any special meaning.

() Parentheses are used to group a *record-data* field that continues to the next line. That is, the end of the line is not recognized within the parentheses.

As stated previously, the value in the *record-type* field controls the format and meaning of the resource record. The detailed content and format for each type of resource record is described in the following:

SOA A **Start of Authority** (SOA) record defines the start of a zone, declares that the current name server is authoritative for the zone, and has the following format:

```
;name        ttl    class  type  data
name                IN     SOA   host person (
                                 serial
                                 refresh
                                 retry
                                 expire
                                 minimum )
```

where:

name Is the name of the domain that defines the top of the zone.

host Indicates the host on which the data file for this zone is stored.

person Specifies the mailbox for the person responsible for the zone. The format is similar to a mailing address, but the at sign (@) that normally separates the user from the host name is replaced by a period (.).

serial Indicates the version number of the zone file. This field should be incremented any time a change is made to the data for the zone.

refresh Specifies how often, in seconds, a secondary name server should check with the primary name server to see if an update is needed. A suggested value for this field is 1 hour (3600).

retry Indicates how long, in seconds, a secondary name server is to retry after a refresh attempt fails. A suggested value for this field is 10 minutes (600).

expire Sets the upper limit, in seconds, that a secondary name server is to use the data before it expires because it has not been refreshed. This value should be fairly large, and a suggested value is 42 days (3600000).

minimum Specifies the minimum time, in seconds, to use as time-to-live values in resource records. A suggested value is one day (86400).

Each zone should have only one SOA record. A sample SOA record is provided in the **named.data** example on page 4-47.

NS A **Name Server** (NS) record, which lists the name of a host that provides DOMAIN service for a particular domain.

```
;domain          ttl    class  type  data
domain                   IN     NS    server
```

where:

domain Contains the name associated with the resource record.

server Identifies the host that provides name service.

Hosts that provide name service do not have to reside in the **named** domain. There should be one NS record for each server for a domain, and NS records for a domain exist in both the zone that delegates the domain and in the domain itself.

A An **Address** (A) record, which has the following format:

```
;host              ttl    class  type  data
host                       IN     A    address
```

where:

host Contains the name of the host system.

address Specifies the Internet address of the host in dotted decimal form.

For example, an address record might look like:

```
star.abc.aus.ibm.com.        IN     A      192.9.201.3
```

There should be one A record for each address of a host.

If the name server host for a particular domain resides inside the domain, then an A (Address) resource record that specifies the address of the server is required. This address record is only needed in the server delegating the domain, not in the domain itself. If, for example, the server for domain aus.ibm.com was fran.aus.ibm.com, then the NS record and required A record would look like:

```
aus.ibm.com.                IN              NS   fran.aus.ibm.com.
fran.aus.ibm.com.           IN              A    192.9.201.14
```

CNAME A **Canonical Name** (CNAME) record, which is used to create an alias for a given host.

```
;alias             ttl    class  type    data
alias                      IN     CNAME  host
```

where:

alias Specifies the name associated with the resource record.

host Contains the official name of the host.

For example, a host named fran.aus.ibm.com uses the following resource record to establish an alias of plan:

```
plan                        IN     CNAME  fran.aus.ibm.com.
```

There should only be one resource record associated with an alias in a particular class.

Aliases can be useful when a host changes its name, since a CNAME record can be used to allow hosts using the old name to continue to reach the host desired.

HINFO A **Host Information** (HINFO) record, which gives information about a particular host.

```
;host        ttl    class  type    data
host                 ANY    HINFO  hardware software
```

where:

host Contains the name of the host system.

hardware Describes the hardware of this host, usually a manufacturer's name, followed by a dash and a model designation. Also referred to as the machine name.

software Describes the software of this host, usually the name of the operating system. Also referred to as the system name.

The *hardware* and *software* descriptions are separated by one or more spaces. No spaces are allowed within these fields, so hyphens or slashes should be used to distinguish parts of names.

A sample HINFO record is:

```
fran                 ANY    HINFO  IBM-RT/PC AIX
```

WKS A **Well-Known Services** (WKS) record, which lists the Well-Known Services provided by the host, and has the following format:

```
;host              ttl    class  type    data
host                       IN     WKS    address protocol services
```

where:

host Contains the official name of the host.

address Specifies the Internet address of the host in dotted decimal form.

protocol Identifies the protocol that provides the services, commonly TCP or UDP.

services Lists the services that are available at the named host.

A sample WKS record for a host offering the same services on all addresses might look like:

```
fran      IN     WKS     192.9.201.3  tcp  telnet ftp smtp
                 WKS     192.9.201.3  udp  time
```

MX A **Mail Exchanger** (MX) record, which specifies where mail for a domain name should be delivered. An MX record has the following format:

```
;name               ttl    class  type    data
name                          IN    MX     preference    host
```

where:

name Contains the name of the domain whose mail is to be handled.

preference Specifies the order in which a mailer should try multiple MX records when delivering mail. The highest preference is 0.

host Identifies the host to which mail for the named domain should be delivered.

There can be multiple MX records for a single domain, and multiple records for the same name can have the same preference.

A host called `fran.aus.ibm.com` that wants its mail to be delivered through one of two different gateway hosts, in order of preference, might use the following MX records:

```
fran.aus.ibm.com.       IN     MX       10     kelly.aus.ibm.com.
                        IN     MX       20     vince.aus.ibm.com.
```

An entire domain of hosts not connected to the Internet might have their mail sent through a mail gateway that can deliver mail to them. In order to receive mail addressed to any host in the domain, the following MX records might be used:

```
ibm.com.                IN     MX       10     kelly.aus.ibm.com.
*.ibm.com.              IN     MX       20     kelly.aus.ibm.com.
```

Note that an asterisk can be used in the MX record to match any host in the `ibm.com` domain, but not to match `ibm.com.` itself.

IN-ADDR.ARPA
The structure of names in the domain system is set up in a hierarchical fashion, so that the address of a name can be found by tracing down the domain structure, contacting a server for each label in the name. Because of this structure based on name, there is no easy way to translate a host address back into its host name.

In order to allow simple reverse translation, the IN-ADDR.ARPA domain was created. This domain uses addresses of hosts as part of a name that points to the data for that host. The IN-ADDR.ARPA domain provides an index to the resource records of each host based on its address. There are subdomains within the IN-ADDR.ARPA domain for each network, based on network number.

Also, to maintain consistency and natural groupings, the 4 octets of a host number are reversed.

For example, the ARPANET is `net 10`, which means that there is a domain called `10.in-addr.arpa`. Within this domain, there is a PTR resource record at `51.0.0.10.IN-ADDR`, which points to the resource records for the host `sri-nic.arpa` (whose address is `10.0.0.51`). Since the NIC is also on the MILNET (`net 26, address 26.0.0.73`), there is also a PTR resource record at `73.0.0.26.in-addr.arpa` that points to the same resource records for SRI-NIC.ARPA. The format of these special pointers is defined in the following section on PTR resource records, along with the examples for the NIC.

PTR A **Domain Name Pointer** (PTR) record, which allows special names to point to another location in the domain.

;special name	ttl	class	type	data
special name		IN	PTR	*name*

where:

special name Contains the Internet address of the host with the octets in reverse order. If you have not defined the IN-ADDR.ARPA domain in your **named.boot** file, this address must be followed by `.in-addr.arpa`.

name Specifies the name of the host.

PTR resource records are mainly used in IN-ADDR.ARPA records for translation of addresses to names. PTR records should use official host names, not aliases.

For example, host `sri-nic.arpa`, whose addresses are `10.0.0.51` and `26.0.0.73`, would have the following records in the respective zone files for `net 10` and `net 26`:

```
51.0.0.10                  IN    PTR    SRI-NIC.ARPA.
73.0.0.26                  IN    PTR    SRI-NIC.ARPA.
```

For further examples of PTR records, see the **named.rev** example on page 4-50.

GATEWAY PTR

The IN-ADDR domain is also used to locate gateways on a particular network. Gateways have the same kind of PTR resource records as hosts, but they also have other PTR records used to locate them by network number alone. These records have only 1, 2, or 3 octets as part of the name, depending on whether they are class A, B, or C networks, respectively.

The gateway host named `gw`, for example, connects three different networks, one each in class A, B, and C. This gateway has the standard resource records for a host in the `csl.sri.com` zone:

```
gw.csl.sri.com.          IN    A     10.2.0.2
                         IN    A     128.18.1.1
                         IN    A     192.12.33.2
```

In addition, this gateway has one of the following pairs of number-to-name translation pointers and gateway location pointers in each of the three different zones (one for each network). In each example, the number-to-name pointer is listed first, followed by the gateway location pointer.

Class A

```
2.0.2.10.in-addr.arpa.      IN    PTR    gw.csl.sri.com.
10.in-addr.arpa.            IN    PTR    gw.csl.sri.com.
```

Class B

```
1.1.18.128.in-addr.arpa.    IN    PTR    gw.csl.sri.com.
18.128.in-addr.arpa.        IN    PTR    gw.csl.sri.com.
```

Class C

```
1.33.12.192.in-addr.arpa.   IN    PTR    gw.csl.sri.com.
33.12.192.in-addr.arpa.     IN    PTR    gw.csl.sri.com.
```

MB A **Mail Box** (MB) record, which is used to define the host where mail for a specific user should be sent.

```
;name            ttl    class   type    data
user                     IN      MB     host
```

where:

user Specifies the name of the user whose mail is to be handled.

host Contains the official name of the host.

For example, a user named `elizabeth` uses the following resource record to have her mail delivered to host `venus.abc.aus.ibm.com.`

```
elizabeth                IN     MB     venus.abc.aus.ibm.com.
```

User names contained in MB records should be unique within a zone.

MR A **Mail Rename Name** (MR) record, which defines aliases for a user.

```
;name              ttl    class  type    data
alias                      IN     MR      user
```

where:

alias Specifies an alternative name for the official user name.

user Defines the official user name the alias should be associated with. The name listed in this field should have a corresponding MB record.

For example, a user named `robert` uses the following resource record to have mail addressed to his department, `drafting`, delivered to his user ID.

```
drafting                   IN     MR      robert
```

MINFO A **Mailbox Information** (MINFO) record, which creates a mail group for a mailing list.

```
;name              ttl    class  type    data
group                      IN     MINFO   requests   maintainer
```

where:

group Defines the name of the mail group.

requests Determines where to send mail such as requests to be added to the mail group.

maintainer Defines the mailbox that should receive error messages. This field is particularly useful when errors in members' names should be reported to a person other than the sender.

An MINFO resource record is generally associated with one or more MG records, but can be used with an MB record.

MG A **Mail Group Member** (MG) record, which lists the members of a mail group.

```
;name              ttl    class  type    data
group                      IN     MG      member
```

where:

group Defines the name of the mail group. This field is optional.

member Specifies the user name of the group member.

The following example shows the resource records required to set up a sales department mailing list:

```
sales   IN  MINFO  sales-dept          maint.sales.aus.ibm.com.
        IN  MG     thomas.sales.aus.ibm.com.
        IN  MG     clerk1.sales.aus.ibm.com.
        IN  MG     franklin.sales.aus.ibm.com.
        IN  MG     sally.sales.aus.ibm.com.
        IN  MG     leo.sales.aus.ibm.com.
```

There are two types of lines that are allowed in **named** data files but do not define resource records. The first type allows including **named** data from other files and has the format:

`$INCLUDE` *filename*

where the $INCLUDE keyword begins in column one and the *filename* entry is a fully qualified file name. This type of entry allows separating different types of database information into separate files.

The second type of line allows resetting the domain origin in a data file and has the format:

`$ORIGIN` *domain*

where the $ORIGIN keyword begins in column one and the *domain* entry is the name of an Internet domain. This type of entry does not cause data to be loaded into a different zone, but allows data for a given zone to be organized into separate files.

Examples

The following examples portray two domains on two different networks:

- Domain abc.aus.ibm.com, Internet address 192.9.201.*n* (where *n* varies for each system on the network)
- Domain xyz.aus.ibm.com, Internet address 128.114.100.*n* (where *n* varies for each system on the network).

In this sample configuration, both the primary and the secondary name servers are in the abc.aus.ibm.com domain, and host hera is the gateway host.

This section contains examples of several of the **named.*** files. For the corresponding **named.boot** files, see the **named.boot** example on page 4-35.

1. The following are sample entries in a **named.ca** file:

```
; root name servers
.          1          IN    NS    relay.ibm.com.
relay.ibm.com.    3600000    IN    A      129.114.1.2
```

2. The following two files are sample entries from the **named.data** files for hosts venus and allen. (The data files can have any name you choose, as long as that name is defined in the **named.boot** file. In this case, the names are named.abcdata and named.xyzdata.)

The **named** database for host venus is in the following file, called named.abcdata:

```
;OWNER         TTL       CLASS TYPE     RDATA
;local domain server is venus
@   IN   SOA   venus   bob.robert.abc.aus.ibm.com. (
                            1.1     ;serial
                            3600    ;refresh
                            600     ;retry
                            3600000 ;expire
                            86400)  ;minimum
    IN    NS   venus
    IN    NS   kronos
    IN    MX   10       venus.abc.aus.ibm.com.
; address for local loopback
localhost                 IN    A        127.1
; address of machines in the same domain
venus                     IN    A        192.9.201.1
venus                     IN    A        128.114.100.1
venus                     IN    MX       10   venus.aub.aus.ibm.com.
veabc                     IN    CNAME    venus
                          IN    WKS      192.9.201.1 udp (tftp
                                         nameserver domain)
                          IN    WKS      192.9.201.1 tcp (echo telnet
                                         smtp discard uucp-path systat
                                         daytime netstat chargen ftp
                                         time whois finger hostnames
                                         domain)
; delimiter for WKS
kronos                    IN    A        192.9.201.2
krabc                     IN    CNAME    kronos
hera                      IN    A        192.9.201.5
                          IN    A        128.114.100.5
robert                    IN    A        192.9.201.6
ernie                     IN    A        192.9.201.7
                          IN    HINFO    IBM-RT/PC AIX
robert.abc.aus.ibm.com.   IN    MX 10    venus.abc.aus.ibm.com.
```

The **named** database for host allen is in the following file, called named.xyzdata:

```
;OWNER              TTL          CLASS TYPE      RDATA
;local domain server is venus
xyz.aus.ibm.com.   IN   SOA   venus.abc.aus.ibm.com.   bob.robe
rt.abc.aus.ibm.com. (
                                  1.1      ;serial
                                  3600     ;refresh
                                  600      ;retry
                                  3600000  ;expire
                                  86400)   ;minimum
xyz.aus.ibm.com.   IN   NS    venus.abc.aus.ibm.com.
                   IN   NS    kronos.abc.aus.ibm.com.
                   IN   MX    10      venus.abc.aus.ibm.com.
; address for local loopback
localhost                    IN    A        127.1
; address of machines in the same domain
allen                        IN    A        128.114.100.3
allen                        IN    MX       10   venus.aub.aus.ibm.com.
alxyz                        IN    CNAME    allen
                             IN    WKS      128.114.100.3 udp (tftp
                                            nameserver domain)
                             IN    WKS      128.114.100.3 tcp (echo telnet
                                            smtp discard uucp-path systat
                                            daytime netstat chargen ftp
                                            time whois finger hostnames
                                            domain)
; delimiter for WKS
fred                         IN    A        128.114.100.10
mike                         IN    A        128.114.100.11
                             IN    HINFO    IBM-RT/PC AIX
```

3. The following is a sample **named.local** file that might appear on hosts in either of the sample domains:

```
@   IN   SOA   venus.abc.aus.ibm.com. gail.zeus.abc.aus.ibm.com.  (
                                 1.1     ;serial
                                 3600    ;refresh
                                 600     ;retry
                                 3600000 ;expire
                                 86400)  ;minimum
       IN    NS     venus.abc.aus.ibm.com.
1      IN    PTR    localhost.
```

4. The following two files are sample entries from the **named.rev** files for hosts `venus` and `allen`. For this example, the files are called `named.abcrev` and `named.xyzrev`.

   The reverse database for host `venus` is in the following file, called `named.abcrev`:

```
@   IN   SOA   venus.abc.aus.ibm.com. bob.robert.abc.aus.ibm.com.  (
                                 1.1     ;serial
                                 3600    ;refresh
                                 600     ;retry
                                 3600000 ;expire
                                 86400)  ;minimum
9.201.192.in-addr.arpa.    IN    NS     venus.abc.aus.ibm.com.
                           IN    NS     kronos.abc.aus.ibm.com.
                           IN    PTR    hera.abc.aus.ibm.com.
;ABC.AUS.IBM.COM Hosts
1.201.9.192.in-addr.arpa.  IN    PTR    venus.abc.aus.ibm.com.
2.201.9.192.in-addr.arpa.  IN    PTR    kronos.abc.aus.ibm.com.
5.201.9.192.in-addr.arpa.  IN    PTR    hera.abc.aus.ibm.com.
6.201.9.192.in-addr.arpa.  IN    PTR    robert.abc.aus.ibm.com.
7.201.9.192.in-addr.arpa.  IN    PTR    ernie.abc.aus.ibm.com.
```

The reverse database for host `allen` is in the following file, called `named.xyzrev`:

```
@   IN   SOA   venus.abc.aus.ibm.com. bob.robert.abc.aus.ibm.com.  (
                              1.1     ;serial
                              3600    ;refresh
                              600     ;retry
                              3600000 ;expire
                              86400)  ;minimum
100.114.128.in-addr.arpa.       IN    NS     venus.abc.aus.ibm.com.
                                IN    NS     kronos.abc.aus.ibm.com.
                                IN    PTR    hera.abc.aus.ibm.com.
;XYZ.AUS.IBM.COM Hosts
3.100.114.128.in-addr.arpa.     IN    PTR    allen.abc.aus.ibm.com.
10.100.114.128.in-addr.arpa.    IN    PTR    fred.abc.aus.ibm.com.
11.100.114.128.in-addr.arpa.    IN    PTR    mike.abc.aus.ibm.com.
;ABC.AUS.IBM.COM Hosts
1.100.114.128.in-addr.arpa.     IN    PTR    venus.abc.aus.ibm.com.
5.100.114.128.in-addr.arpa.     IN    PTR    hera.abc.aus.ibm.com.
```

Files

/etc/named.*	Files that contain the DOMAIN nameserver database.
/usr/lpp/tcpip/samples/addrs.awk	Sample **awk** script for converting an **/etc/hosts** file to an **/etc/named.rev** file.
/usr/lpp/tcpip/samples/hosts.awk	Sample **awk** script for converting an **/etc/hosts** file to an **/etc/named.data** file.
/usr/lpp/tcpip/samples/named.boot	Sample **named.boot** file.
/usr/lpp/tcpip/samples/named.data	Sample **named.data** file.

Related Information

In this book: "**named**" on page 3-20 and "**named.boot**" on page 4-34.

net

Purpose

Defines adapter cards and IP characteristics for TCP/IP.

Synopsis

/etc/net

Description

The **/etc/net** file contains a keyword associated with each adapter card (device) that TCP/IP can use and a stanza that describes the characteristics for the adapter. There is one stanza entry for each adapter defined for use with TCP/IP. There can also be a stanza with the name **ip_security**, which defines the level of security for all network interfaces for this host. The **netconfig** command processes the **/etc/net** file to establish the connection between the Interface Program and the adapters.

The format of an adapter stanza in **/etc/net** is:

```
sys_stanza_name:
        dialstring =
        disconnect =
        dstaddr =
        inetlen =
        localbroadcast =
        netaddr =
        protocol =
        r_inetlen =
        security =
        subnetmask =
```

The format of the **ip_security** stanza is:

```
ip_security:
        level =
        authority =
```

The information contained in an adapter stanza is:

sys_stanza_name: Starting in column one, the name of the device defined in the **/etc/system** file. (The **/etc/system** file is defined using the **devices** command.) For example, use `net0:` for the Baseband Adapter and `token0` for the IBM Token-Ring Adapter.

dialstring An optional string used for dialing by a tty device.

disconnect A true or false value that indicates whether connections should be dropped after a period of inactivity. A value of **true** indicates that the connection should be dropped if inactive, while a value of **false** indicates that the connections should be maintained even if inactive. This keyword is only valid for serial line interfaces.

dstaddr The IP network address for the destination host of a point-to-point connection, used for serial line interfaces. The address must be specified in dotted decimal notation. If this keyword is not present for a serial line interface, its default value is 0.

inetlen The maximum IP packet length for transmission to this adapter. The default packet size is 1500 bytes for the Baseband Adapter and 1568 bytes for the IBM Token-Ring Adapter. The default packet size for 802.3 Adapter is 1500 bytes. The default packet size for the X.25 Adapter is 1007 bytes. The maximum frame size that can be passed through a Token-Ring bridge is 1K bytes. If the frame size is 1K bytes, **inetlen** should be set to 576. If the frame size is larger, **inetlen** can be set to a larger value.

Warning: For X.25 interfaces, the value of **inetlen** must be less than or equal to the size of the VRM buffer specified in **devices** or unpredictable results may occur.

localbroadcast An optional keyword applicable to Token-Ring devices only. Used to indicate if machines on the same Token-Ring Network, but different rings, can communicate. The values can be:

- **true** - Only machines on the same ring can communicate.
- **false** - Machines on same network but different rings can communicate.

If not specified, the default value is **true**.

netaddr The IP network address to be used for this adapter. The address must be specified in dotted decimal notation. This is a required keyword for all interfaces.

protocol A value used to specify whether an 802.3 interface is being described. The value **802.3** indicates an 802.3 interface, while the value **standard** indicates standard Ethernet headers. If this keyword is present, its default value for interfaces named **net#** is **standard**.

r_inetlen An optional, maximum IP packet length for remote transmission (transmission outside of the local network). If not specified, default is 576 bytes.

subnetmask An optional, hexadecimal mask that defines a sub-network within a local address. A mask is specified only for the local address portion of an Internet address. If **subnetmask** is not set, sub-networks are not used. If sub-networks are to be used, all hosts on the network must have **subnetmask** set.

The information contained in an **ip_security** stanza is:

level Specifies the security level as one of the following values, listed in order from least to most secure:

- None
- Unclassified
- Confidential
- Secret
- Top_Secret.

When the level of security is defined as **None** or **Unclassified**, no IP Option header is added to the IP header.

authority Defines the authority who determines the remainder of the option definition for that value from the following list:

- GENSER
- SIOP
- DSCCS-SPINTCOM
- DSCCS-CRITICOM

More than one authority can be specified, in which case the values are separated by commas but no spaces. For example:

```
authority = GENSER,SIOP,DSCCS-SPINTCOM
```

Examples

Following are sample entries in the **/etc/net** file:

1. The following is a stanza for a Baseband Adapter using an IP address of 192.100.10.1. In this example, which uses a class C IP address, the four high-order bits of the local address represent the subnet number, while the four low-order bits refer to the local host on the sub-network. The maximum size of packets transmitted outside the local network is 1500 bytes. No security option is specified for this example host, so any IP datagram can be sent or received through all interfaces for this host. IP headers for this host do not contain the IP security option.

```
net0:
        netaddr = 192.100.10.1
        inetlen = 1500
        subnetmask = FFFFFF00
        r_inetlen = 1500
```

2. The following is a stanza for an IBM Token-Ring Adapter using an IP address of 128.114.100.1. This interface can communicate with all rings on its Token-Ring network. Because **r_inetlen** is not specified, the maximum size of packets transmitted outside the local network is 576 bytes. The IP security option for this host is specified as **level = confidential**, so any IP datagram sent or received by this host must be to or from a host at the same level. IP headers for this host contain the IP security option.

```
token0:
        netaddr = 128.114.100.1
        inetlen = 1568
        localbroadcast = false

ip_security:
        level = confidential
        authority = SIOP
```

3. The following is a stanza for a 802.3 Adapter using an IP address of 192.100.11.1. The maximum size of packets transmitted outside the local network is 1500 bytes.

```
net1:
        netaddr = 192.100.11.1
        inetlen = 1500
        r_inetlen = 1500
        protocol = 802.3
```

4. The following is a stanza for a X.25 Adapter using an IP address of 128.214.101.1. The maximum size of packets transmitted to the local X.25 network is 1007 bytes.

```
X25a0:
        netaddr = 128.214.101.1
        inetlen = 1007
```

5. The following is a stanza for a tty device using an IP address of 192.117.102.1. The maximum size of packets transmitted through this tty device is 576 bytes. The device connects to a remote host by sending the dial string ATDT5550000 to a modem, and connections to the device are automatically dropped after a period of inactivity. The IP security option for this host is specified as **level = none**, so any IP datagram can be sent or received through this interface and any other interfaces for this host. IP headers for this host do not contain the IP security option.

```
tty0:
        netaddr = 192.117.102.1
        inetlen = 576
        disconnect = true
        dialstring = ATDT5550000

ip_security:
        level = none
```

6. Use the following **/etc/net** stanzas to establish a serial line interface on two hosts that are connected by a serial cable. These stanzas establish a point-to-point connection, and the connection will not be dropped even after a period of inactivity.

   a. On the first host, `engineer`, add the following stanza:

   ```
   tty1:
           netaddr = 192.9.201.3
           dstaddr = 192.9.354.7
           disconnect = false
           inetlen = 2048
           r_inetlen = 2048
   ```

   b. On the second host, `market`, add the following stanza:

   ```
   tty1:
           netaddr = 192.9.354.7
           dstaddr = 192.9.201.3
           disconnect = false
           inetlen = 2048
           r_inetlen = 2048
   ```

File

/etc/net Defines adapter cards for TCP/IP.

Related Information

In this book: "**netconfig**" on page 2-35.

networks

Purpose

Contains the network name database.

Synopsis

/etc/networks

Description

The **networks** file contains information about the known networks that comprise the DARPA Internet. Each network is represented by a single line in the **networks** file. The format for the entries in **networks** is:

name number aliases

where:

name	is the official network name.
number	is the network number.
aliases	are the unofficial names used for the network.

Items on a line are separated by one or more blanks or tab characters. Comments begin with the # character, and routines that search **networks** do not interpret characters from the beginning of a comment to the end of that line. Network numbers are specified in dotted decimal notation. A network name can contain any printable character except a field delimiter, new line character, or comment character.

The **networks** file is normally created from the official network database maintained at the Network Information Control Center (NIC). The file may need to be modified locally to include unofficial aliases or unknown networks.

Files

/etc/networks	Contains the network name database.
/usr/lpp/tcpip/samples/networks	Sample **/etc/networks** file.

Related Information

In this book: "**routed**" on page 3-28.

The **getnetent,...** routines in *AIX Operating System Technical Reference*.

protocols

Purpose

Defines the Internet protocols used on the local host.

Synopsis

/etc/protocols

Description

This file contains information about the known protocols used in the DARPA Internet. Each protocol is represented by a single line in the **protocols** file. Each entry is of the form:

name number aliases

where:

name is the official Internet protocol name.

number is the protocol number.

aliases are the unofficial names used for the protocol.

Items on a line are separated by one or more blanks or tab characters. Comments begin with the # character, and routines that search the **protocols** file do not interpret characters from the beginning of a comment to the end of the line. A protocol name can contain any printable character except a field delimiter, new line character, or comment character.

File

/etc/protocols Defines Internet protocols for local host.

Related Information

The **getprotoent,...** routines in *AIX Operating System Technical Reference.*

rc.tcpip

Purpose

Sets up network interfaces, defines the local IP host name and address, initializes Internet routes, and starts the Interface Program daemons.

Synopsis

/etc/rc.tcpip

Description

The **rc.tcpip** file contains the commands necessary for a host to run the Interface Program. The **netconfig** command defines the interfaces (adapter cards) to be configured. The **hostname** command defines the host name and address. The parameter in **rc.tcpip** for **/bin/hostname** must be a name for your system that can be resolved by an entry in **/etc/hosts** to an address in **/etc/net**. The **route** command sets the routing tables. The remaining entries execute some Interface Program daemons and other daemons that are not a part of Interface Program but depend on it:

gated	**routed**
inetd	**rwhod**
lpd	**sendmail**
named	**syslogd**
portmap	**uucpd**

If you need to start a daemon that is not automatically started in your TCP/IP initialization, simply edit the **/etc/rc.tcpip** file and uncomment the line for that daemon.

Note: The **lpd** daemon should run only on hosts that act as print servers. The **named** daemon should run only on hosts that function as DOMAIN name servers.

For more information on each of these daemons, see Chapter 3, "Server Commands" on page 3-1.

If the Interface Program is to be initialized when the system is started, the following line must be uncommented in the **/etc/rc.include** command file, which is started by an entry in the **/etc/rc** file:

```
sh /etc/rc.tcpip
```

Example

The following example section of an **rc.tcpip** file shows how to define **zeus** as the host name for this system and to establish a default gateway:

```
# Set up local Internet Address

/bin/hostname zeus.abc.aus.ibm.com     >/dev/console
# Set up default gateway
# Add route commands here
route add 0 192.9.201.1
```

Files

/bin/hostname	Sets the name for a host.
/etc/netconfig	Configures the Interface Program for multiple adapter cards.
/etc/rc.include	Performs part of system initialization.
/etc/rc.tcpip	Performs TCP/IP startup functions.

resolv.conf

Purpose

Defines DOMAIN nameserver information for local resolver routines.

Synopsis

/etc/resolv.conf

Description

If this file exists, the local resolver routines either use a local name resolution database maintained by a local **named** process to resolve Internet names and addresses, or they use the Domain Name Protocol (DOMAIN protocol) to request name resolution services from a remote DOMAIN nameserver host. In either case, if the name resolution information is unavailable, the routines then attempt to use the **/etc/hosts** file for name resolution.

Note: If the **/etc/resolv.conf** file does not exist, the resolver routines attempt name resolution using the local **/etc/hosts** file. If you are not using a DOMAIN name server, this file should not exist.

On DOMAIN nameserver hosts using the **named** server, *this file must exist and must be empty.* On other hosts, this file contains at most one `domain` entry and between one and **MAXNS**, which is defined as 10, `nameserver` entries.

A `domain` entry tells the resolver routines the default domain name to append to names that do not end with a period (.). This entry is of the form:

`domain` *domainname*

where *domainname* is the name of the local Internet domain. If there is no `domain` entry in the file, the default domain is the domain returned by the **gethostname** routine (that is, everything following the first period). If the host name does not have a domain name included, the root (.) domain is assumed.

A `nameserver` entry tells the resolver routines the Internet address of a remote DOMAIN name server for the local domain. This entry is of the form:

`nameserver` *address*

where *address* is the dotted decimal address of the remote name server. If more than one name server is listed, the resolver routines query each name server (in the order listed) until either the query succeeds or the maximum number of attempts have been made.

Each line in the **resolv.conf** file must begin with either `domain` or `nameserver`, followed by blanks or tabs and a corresponding *domainname* or *address*.

Examples

The following are sample entries in the **/etc/resolv.conf** file for a host that is not a name server. (The nameserver hosts should have an empty **resolv.conf** file.)

```
domain abc.aus.ibm.com
nameserver 192.9.201.1
nameserver 192.9.201.2
```

Files

/etc/resolv.conf	Defines nameserver information for resolver routines.
/usr/lpp/tcpip/samples/resolv.conf	Sample **resolv.conf** file.

Related Information

In this book: "**named**" on page 3-20.

The **gethostbyaddr,...** and **gethostname,...** routines in *AIX Operating System Technical Reference.*

services

Purpose

Defines the sockets and protocols used for Internet services.

Synopsis

/etc/services

Description

This file contains information about the known services used in the DARPA Internet. Each service is listed in this file on a single line of the form:

servname portnum/protname aliases

where:

servname is the official Internet service name.

portnum is the socket port number used for the service.

protname is the transport protocol used for the service.

aliases is a list of unofficial service names.

Items on a line are separated by blanks or tabs. Comments begin with a # character and continue to the end of the line.

Examples

The following are sample entries in the **/etc/services** file for the **inetd** internal services:

```
echo         7/tcp
echo         7/udp
discard      9/tcp      sink null
discard      9/udp      sink null
daytime     13/tcp
daytime     13/udp
chargen     19/tcp      ttytst source
chargen     19/tcp      ttytst source
ftp         21/tcp
time        37/tcp      timeserver
time        37/udp      timeserver
```

File

/etc/services Defines sockets and protocols used for Internet services.

Related Information

The **getservent,...** routines in *AIX Operating System Technical Reference.*

.netrc

Purpose

Specifies automatic login information for **ftp** and **rexec**.

Synopsis

$HOME/.netrc

Description

The **.netrc** file contains the information used by the automatic login (auto-login) feature of the **rexec** and **ftp** commands. It is a hidden file in a user's home directory and must be owned by the user executing the command or by `root`. If the **.netrc** file contains a login password, the file's permissions must be set to 600 (read and write by owner only).

Note: The **.netrc** file is not used by any programs when your system is in controlled access mode.

The **.netrc** can contain the following entries (separated by blanks, tabs, or new-lines):

machine *hostname*
: where *hostname* is the name of a remote host. This entry begins the definition of the auto-login process for the specified host. All following entries up to the next `machine` entry or the end of the file apply to that host.

login *username*
: where *username* is the user name to use at the remote host. If this entry is found, the auto-login process initiates a login using the specified name. If this entry is missing, the auto-login process fails.

password *password*
: where *password* is the login password to be used. The auto-login process supplies this password to the remote server. A login password must be established at the remote host and that password must be entered in this file, or the auto-login process fails and the user is prompted for the login password.

account *password*
: where *password* is the account password to be used. If this entry is found and an account password is required at the remote host, the auto-login process supplies the password to the remote server. If the remote host requires an account password but this entry is missing, the auto-login process prompts for the account password.

macdef *macroname*

where *macroname* is the name of an **ftp** subcommand macro. (See "**ftp**, **xftp**" on page 2-9.) The macro is defined to contain all of the following **ftp** subcommands up to the next blank line or the end of the file. If the macro is named `init`, **ftp** executes the macro upon successful completion of the auto-login process. The **rexec** command does not recognize a `macdef` entry.

The **/usr/lpp/tcpip/samples/netrc** file is a sample **.netrc** file. Use the following procedure to create a **.netrc** file based on the sample:

1. Copy **/usr/lpp/tcpip/samples/netrc** to your home directory.
2. Edit **$HOME/netrc** to supply the appropriate *hostname*, *username*, and *password*.
3. Set the permissions on **$HOME/netrc** to 600.
4. Rename the file **.netrc** (the initial ., or dot, causes the file to be hidden).

Files

$HOME/.netrc	Contains automatic login information.
/usr/lpp/tcpip/samples/netrc	Sample **.netrc** file.

Related Information

In this book: "**rexec**" on page 2-51 and "**ftp**, **xftp**" on page 2-9.

.rhosts

Purpose

Specifies remote users that can use a local user account.

Synopsis

$HOME/.rhosts

Description

This file contains a list of remote users who are not required to supply a login password when they execute the **rcp**, **rlogin**, and **rsh** commands using a local user account. This file is a hidden file in the local user's home directory and must be owned by the local user or local `root`. The permissions of the **.rhosts** file should be set to 600 (read and write by owner only). Each entry in the file is of the form:

host username

where *host* is the name of the remote host and *username* is the login name of the remote user. If the remote host is in a different domain than the local host, the full domain name must be specified.

Examples

The following sample entries in the **/u/amy/.rhosts** file on host `zeus` allow users `mark` and `geo` at remote host `venus` to log in to user `amy`'s home directory on host `zeus`:

```
venus geo
venus mark
```

File

$HOME/.rhosts Specifies remote users that can use a local user account.

Related Information

In this book: "**rlogin**" on page 2-54 and "**rsh, remsh**" on page 2-60.

.3270keys

Purpose

Defines a user keyboard mapping and colors for TELNET (3270).

Synopsis

$HOME/.3270keys

Description

The **.3270keys** file allows a user to have a TELNET (3270) key mapping that is different from the default mapping. If you are using a color display, you can also change this file to customize the colors for various 3270 display attributes. (The default mapping in the **/etc/3270.keys** file is generic; the **.3270keys** file allows users to tailor key mappings to the RT keyboard and to select alternate colors for 3270 display attributes.) The **.3270keys** file exists as a hidden file (the name begins with a **.**, or dot) in the user's home directory.

Note: When remapping keys to customize your **.3270keys** file, keep in mind that you cannot map a 3270 function to the **ESC** key alone. You can specify the **ESC** key only in combination with another key.

The **/usr/lpp/tcpip/samples/3270keys.rt** file is a sample that can be used to create a **.3270keys** file. To create a **$HOME/.3270keys** file, use the following procedure:

1. Copy **/usr/lpp/tcpip/samples/3270keys.rt** to the name **.3270keys** in your home directory.
2. Edit the **local key seq** column of **.3270keys** as necessary to create the new mapping. Following is a general procedure for modifying **.3270keys**:

   Note: The specific methods for performing these steps may be different for different editors. If you have questions about how your editor performs a task, consult the documentation provided with the program.

   a. Open the **.3270keys** file with an editor.
   b. Move the cursor to the **local key seq** column and then to the 3270 key that you want to map.
   c. Put the editor in **insert** mode (in the **vi** editor, use the **i** subcommand).
   d. Press the key or key sequence that allows you to enter nonprinting characters (in the **vi** editor, press **Ctrl-V**).

e. Press the key that is to be mapped to the **3270** key.

f. End **insert** mode (in the **vi** editor, enter **ESC**).

g. Repeat the steps for each key that is to be mapped.

3. If you are using a color display and want to change the colors used for 3270 display attributes, follow these steps while in an editor:

   a. Move the cursor to the section of the file where the colors are defined.

   b. Locate the 3270 attribute you want to change.

   c. Replace the default color for that attribute with the color of your choice.

   d. Repeat the two preceding steps for each attribute you want to change.

   For example, to change the low intensity, unprotected fields on the display from the default color of green to magenta, locate the line for the `low_unprot` attribute and replace `green` with `magenta`.

   Valid colors for the attributes are listed in the file.

4. When you are satisfied with the key mappings and color definitions, save the **.3270keys** file and stop the editor.

Note: You can also change the default key mappings by editing **/etc/3270keys.rt** for RT mapping and **/etc/3270.keys** for non-RT mapping.

Files

/etc/3270.keys	Default keyboard mapping for non-RT keyboards.
/etc/3270keys.rt	Default keyboard mapping for RT keyboard.
/usr/lpp/tcpip/samples/3270keys.rt	Sample RT keyboard mapping.
$HOME/.3270keys	Defines user's keyboard mapping for TELNET.

Related Information

In this book: "**telnet, tn**" on page 2-72.

Appendix A. Customizing the Program

About This Appendix

Before you can use the Interface Program, you must install the necessary hardware and software, and then customize the software to meet your requirements. For information about the software and hardware prerequisites, see "Before You Begin" on page iv. Once you have installed the Interface Program, return to this section for information about establishing a network.

How to Customize an Interface Program Network

Following are the general steps in customizing an Interface Program network. You must be able to supply the specific information pertaining to your system and network. In addition to the information contained here, you may find it helpful to look at the sample files provided in the **/usr/lpp/tcpip/samples** directory.

1. Set the jumpers on the adapter or adapters you are using (the IBM RT Baseband Adapter for use with Ethernet or with 802.3, the IBM Token-Ring Network RT Adapter, or a customer-supplied X.25 Adapter or 802.3 Adapter) according to the instructions in *Options Installation* or provided by the manufacturer.
   - If more than one Baseband Adapter is installed, each one must be set at a different interrupt level and address space. Interrupt levels cannot be shared among Baseband Adapter cards or with other devices.
   - Multiple Token-Ring Adapters can share the same interrupt level.
   - If more than one 802.3 Adapter is installed, each must be set at a different interrupt level and address space. Interrupt levels cannot be shared among 802.3 Adapters or with other devices.
   - The X.25 Adapter cannot share interrupt levels with other devices. Only one X.25 Adapter is supported.

2. Use the **devices** command to add adapters and generic data links. You can add Baseband Adapters and IBM Token-Ring Adapters to run the Interface Program. The keyword values must match the adapter switch settings.

3. Use the **devices** command to add ttys. If you plan to use a serial line interface, the tty should be set for 8 bits and no parity, and you should use the highest baud rate supported by both ends of the serial connection. For more information on the **devices** command, see *AIX Operating System Commands Reference*. For more information on communications by modem, see *Managing the AIX Operating System.*

4. Use the **devices** command to add ptys if your system is to be available to other users for remote login (as in TELNET sessions) or for socket-based commands such as **rlogin**. Select the appropriate terminal type for each pty, just as you would for a tty. (**hft** is generally used as the terminal type for RT-to-RT connections.)
   - If the pty is to be used for TELNET (remote login to this host), set the values of **logger** and **automatic enable** to **true**. (**automatic enable** causes the **penable** command to be run for that device at system start, making it available for use by TELNET.)
   - If the pty is to be used by **rlogin** or **rsh** with no command specified, set the values of **logger** and **automatic enable** to **false**. You must have three ptys for each of these commands running on your system at any given time.

For more information about customizing ptys for TELNET, see "**telnet, tn**" on page 2-72. For information about customizing ptys for **rlogin**, see "**rlogin**" on page 2-54.

5. Set the value for the device characteristic, **rdto** (Receive Data Transfer Offset) when using the **devices** command to install an Ethernet, Token-Ring, X.25, or 802.3 device. **rdto** specifies the byte offset pad within a communication buffer. This byte offset pad allows received frames to have optional field alignment. Set this field to the following value for the type of device you are installing:

```
Ethernet      26
Token-Ring     0
X.25          40
802.3         26
```

6. Customize the **/etc/hosts** file according to the information under "**hosts**" on page 4-24.

7. Customize the **/etc/rc.tcpip** file according to the information under "**rc.tcpip**" on page 4-62. Remember that you must use the same name as the parameter to **/bin/hostname** in **/etc/rc.tcpip** that you use as the name of the local host in the **/etc/hosts** file, and it must correspond with the appropriate address in the **/etc/net** stanza. If you use **sendmail** in a DOMAIN environment, this name must be the full domain name of the host. Customizing this file may include uncommenting some daemons in addition to those already uncommented.

   Note: The name of your host must be used as the parameter to **/bin/hostname** in **/etc/rc.tcpip**, and it must correspond with the appropriate address in the **/etc/net** stanza.

8. Customize the **/etc/net** file according to the information under "**net**" on page 4-52.

Step 9 on page A-3 to step 19 on page A-5 are optional and perform further customization of the network. Continue with step 20 on page A-5 to initialize a basic network.

9. Customize the **$HOME/.netrc** file, which contains automatic login information used by the **ftp** and **rexec** commands, according to the information under "**.netrc**" on page 4-68.

10. Customize the **$HOME/.rhosts** file, which is used by the **rcp**, **rlogin**, and **rsh** commands, according to the information under "**.rhosts**" on page 4-70.

11. Edit the **/etc/master** file, and locate the **sysparms:** stanza. Change the value of the **procs** and **kprocs** keywords as follows:

    - For each Ethernet, X.25, and 802.3 interface defined in **/etc/net**, add 1 to each keyword value.
    - For each Token-Ring interface defined in **/etc/net**, add 2 to each keyword value.

In order for these changed values to take effect, you must rebuild the kernel. For instructions on how to generate a new kernel, see *Managing the AIX Operating System.*

12. Customize the **/etc/hosts.equiv** file, which is used by the **lpd**, **rcp**, **rlogin**, and **rsh** commands, according to the information under "**hosts.equiv**" on page 4-28.

13. If you want to establish a remote print server, customize the **/etc/hosts.lpd** file on the print server host according to the information under "**hosts.lpd**" on page 4-30.

14. On each host in the network that plans to use the print server, do the following:

    a. Add a **printserver** entry to the **/etc/hosts** file according to the instructions under "**hosts**" on page 4-24.

    b. Run the shell script **prtsvr.inst** to install the necessary stanzas in the **/etc/qconfig** file. **prtsvr.inst** can be found in the **/usr/lpp/tcpip/samples** directory. To print remotely, send jobs to the queue **rp0**.

15. If you plan to use the **telnet** command and want to define your own key mappings or colors for a color display, perform these steps:

    a. Create and customize a **$HOME/.3270keys** file according to the information under "**.3270keys**" on page 4-72.

    b. Customize the **/etc/3270keys.rt** and **/etc/3270.keys** files if necessary. For related information, see "**.3270keys**" on page 4-72.

16. Run **/etc/routed** or **/etc/gated** on all hosts that function as gateways. The **routed** or **gated** command can also be run on non-gateway machines. To run **/etc/routed** or **/etc/gated** automatically when the system is started, remove the comment symbol from the beginning of the line for that daemon in the **/etc/rc.tcpip** file.

    Note: Running both the **routed** and the **gated** servers on the same host may produce unpredictable results. Choose the daemon most suited to your needs and uncomment only that command in **/etc/rc.tcpip**.

    - The **routed** command provides routing information for RIP packets. **routed** refers to the **/etc/gateways** and the **/etc/networks** files, if they exist, for a broader view of the networks. Samples of **/etc/gateways** and **/etc/networks** are located in the **/usr/lpp/tcpip/samples** directory.

        a. Customize the **/etc/networks** file according to the information under "**networks**" on page 4-58.

        b. Customize the **/etc/gateways** file according to the information under "**gateways**" on page 4-21.

- The **gated** daemon provides routing for RIP, EGP, and HELLO packets. **gated** gets its configuration information from the **/etc/gated.conf** file. Customize the **/etc/gated.conf** file according to the information under "**gated.conf**" on page 4-7.

17. If your host is to act as a DOMAIN name server, follow these steps:

    a. Remove the comment symbol from the line that starts the domain name daemon (**/usr/lpp/tcpip/etc/named**) in the **rc.tcpip** file. For more information about setting up the domain name server, see "**named**" on page 3-20.

    b. Use the **addrs.awk** and **hosts.awk** scripts located in the **/usr/lpp/tcpip/samples** directory to convert your existing **/etc/hosts** file to a DOMAIN database.

18. If you are using a DOMAIN name server on a remote host, customize the **/etc/resolv.conf** file according to the instructions under "**resolv.conf**" on page 4-64.

19. If the host is not a time server but needs to have access to one (when using the **setclock** command), add a **timeserver** entry to the **/etc/hosts** file according to the instructions under "**hosts**" on page 4-24.

20. If the Interface Program is to be initialized when the system is started, uncomment the following line in the **/etc/rc.include** command file, which is started by an entry in the **/etc/rc** file:

    ```
    sh /etc/rc.tcpip
    ```

    (This causes the network to be configured and the daemons started when the system is started.)

21. If you plan to operate your system in controlled access mode, issue the **securetcpip** command from a shell. This command is provided in the form of a shell script (see the **sh** command in *AIX Operating System Commands Reference* for more details) so that you can customize it to suit your specific installation. For more information on the **securetcpip** command, see "**securetcpip**" on page 2-67.

22. Run **shutdown** and then restart the system.

Appendix B. Examples

About This Appendix

This appendix contains examples of how to use the sockets interface to the Internet protocols. If you want to try these programs on your system, you must use a text editor and enter them exactly as shown. Once you have entered the code, you can compile and run them as you would any other program.

To understand these examples, you must be familiar with the C programming language.

Running the inetd Test Program

The **inetdtst** program is provided as an example of a socket-based TCP/IP program.

To compile the **inetdtst** program, enter the following command:

```
cc -o inetdtst -D BSD_INCLUDES inetdtst.c
```

Before running the test program for **inetd**, be sure the following conditions are met:

1. TCP/IP must be installed, properly configured, and running.
2. The **/etc/inetd.conf** file must be installed.
3. The **inetd** daemon must be installed and running. For more information on starting this daemon, see "**inetd**" on page 3-13.
4. Entering `./inetdtst` in the directory where **inetdtst.c** is stored gives a usage message.

inetdtst.c

```
/*
 * inetdtst.c - test of inetd built in functions
 */
#include <sys/param.h>
#include <sys/socket.h>
#include <time.h>

#include <netinet/in.h>
#include <arpa/inet.h>

#include <errno.h>
#include <stdio.h>
#include <netdb.h>

extern  int errno;

int     debug = 0;

int echo_stream(), discard_stream(), machtime_stream();
int daytime_stream(), chargen_stream();
int echo_dg(), discard_dg(), machtime_dg(), daytime_dg(), chargen_dg();
```

```
struct biltin {
        char    *bi_service;            /* internally provided service name */
        int     bi_socktype;            /* type of socket supported */
        short   bi_fork;                /* 1 if should fork before call */
        short   bi_wait;                /* 1 if should wait for child */
        int     (*bi_fn)();             /* function which performs it */
} biltins[] = {
        /* Echo received data */
        "echo",         SOCK_STREAM,    1, 0,   echo_stream,
        "echo",         SOCK_DGRAM,     0, 0,   echo_dg,

        /* Internet /dev/null */
        "discard",      SOCK_STREAM,    1, 0,   discard_stream,
        "discard",      SOCK_DGRAM,     0, 0,   discard_dg,

        /* Return 32 bit time since 1970 */
        "time",         SOCK_STREAM,    0, 0,   machtime_stream,
        "time",         SOCK_DGRAM,     0, 0,   machtime_dg,

        /* Return human-readable time */
        "daytime",      SOCK_STREAM,    0, 0,   daytime_stream,
        "daytime",      SOCK_DGRAM,     0, 0,   daytime_dg,

        /* Familiar character generator */
        "chargen",      SOCK_STREAM,    1, 0,   chargen_stream,
        "chargen",      SOCK_DGRAM,     0, 0,   chargen_dg,
        0
};

char *myname;
char hnamebuf[ 256 ];
struct sockaddr_in sin;

main(argc, argv, envp)
int argc;
char *argv[];
char *envp[];
{
```

```
register struct biltin *bi;
register struct servent *sp;
register struct hostent *hp;
struct in_addr hostaddr;
int     fd;                     /* socket's file descriptor */
char    *hostname;
char    *service;
int     socktype;

char    *cp;
char    buf[4096];

myname = argv[0];
if( (argc < 3) || (argc > 4) )
        usage("arg count");

if( argc == 4 ) {
        /* were given a hostname to use */
        hostname = argv[1];
        argv++;
} else {
        /* no hostname given, use ourself */
        if( gethostname( hnamebuf, sizeof(hnamebuf) ) != 0 ) {
                perror("couldn't get name of self via gethostname");
                exit(1);
        }
        hostname = hnamebuf;
}

service = argv[1];
if( strcmp( argv[2], "tcp" ) == 0 )
        socktype = SOCK_STREAM;
else
if( strcmp( argv[2], "udp" ) == 0 )
        socktype = SOCK_DGRAM;
else {
        usage("protocol not udp or tcp");
}
```

```
for (bi = biltins; bi->bi_service; bi++)
        if (bi->bi_socktype == socktype &&
            strcmp(bi->bi_service, service) == 0)
                break;
if (bi->bi_service == 0) {
        sprintf( buf, "service \"%s\" not found in list of built-ins\n",
            service );
        usage( buf );
}

hp = gethostbyname( hostname );
if (hp == 0) {
        sprintf( buf, "host \"%s\" not found by gethostbyname\n",
            service );
        usage( buf );
}
bcopy( hp->h_addr_list[0], &hostaddr, sizeof(struct in_addr) );
printf("address of host \"%s\" is 0x%x\n", hostname, hostaddr );

sp = getservbyname( service, argv[2] );
if (sp == 0) {
        sprintf( buf, "service \"%s\" not found by getservbyname\n",
            service );
        usage( buf );
}


/* found "valid" service and socket type, create a socket */
if (( fd = socket(AF_INET, socktype, 0)) < 0) {
        perror(" socket create failed " );
        exit(1);
}

sin.sin_family = AF_INET;
sin.sin_port = sp->s_port;
sin.sin_addr = hostaddr;
```

```
        if(    (connect( fd, &sin, sizeof(sin) ) != 0) ) {
                perror(" socket connect failed " );
                exit(1);
        }

        (*bi->bi_fn)( fd );     /* invoke function to do test */

        fprintf( stderr, "%s: all done, bye\n", myname );
        exit(0);
}

/*
 * the tests of the Internet services provided internally by inetd:
 */

echo_stream( s )                /* Echo service -- echo data back */
int s;
{
        char buffer[BUFSIZ];
        int i;

        printf(" type data to be echoed\n" );

        while( (gets( buffer ) == buffer)  ) {
                i = strlen(buffer);
                buffer[i] = '\n';
                i++;
                if( write(s, buffer, i ) != i ) {
                        break;
                }
                if( read( s, buffer, i ) != i ) {
                        printf("echo read err\n");
                        break;
                }
                buffer[i] = '\0';
                printf("ECHO: %s\n", buffer );
        }
```

```
        exit(0);
}

echo_dg( s )                    /* Echo service -- echo data back */
int s;
{
        char buffer[BUFSIZ];
        int i;

        printf(" type data to be echoed\n" );

        while( (gets( buffer ) == buffer)  ) {
                i = strlen(buffer);
                buffer[i] = '\n';
                i++;
                if( write( s, buffer, i ) != i ) {
                        break;
                }
                if( read( s, buffer, i ) != i ) {
                        printf("echo read err\n");
                        break;
                }
                buffer[i] = '\0';
                printf("ECHO: %s\n", buffer );
        }
        exit(0);
}

discard_stream( s )             /* Discard service -- ignore data */
int s;
{
        long    datasize;
        time_t  starttime;
        time_t  endtime;
        char buffer[BUFSIZ];
        register long i;
        register int sz;
```

```
	printf("how much data to write to discard? :" );
	fflush(stdout);
	if( gets( buffer ) != buffer ) {
		printf("input error???, bye\n");
		exit(1);
	}
	datasize = strtol( buffer, (char **)NULL, 0 );
	printf("writeing %d bytes to discard\n", datasize );

	sz = BUFSIZ;
	i = datasize;
	starttime = time( (long *)0 );
	while( i > 0 ) {
		if( i < BUFSIZ )
			sz = (int)i;
		if( write( s, buffer, sz ) != sz ) {
			perror("discard write failed");
			exit(1);
		}
		i -= sz;
	}
	endtime = time( (long *)0 );

	i =  endtime - starttime;
	if( i <= 0 ) {
		printf("wrote %d bytes in < 1 second\n", datasize );
		exit(0);
	}
	printf("wrote %d bytes in %d seconds\n", datasize, i );
	printf("   (%d Kbytes per second )\n",
	    (datasize/(i))/1024 );
	exit(0);
}

discard_dg( s )		/* Discard service -- ignore data */
int s;
{
	long	datasize;
```

```
time_t  starttime;
time_t  endtime;
char buffer[BUFSIZ];
register long i;
register int pktsz;
register int sz;

printf("how much data to write to discard? :" );
fflush(stdout);
if( gets( buffer ) != buffer ) {
        printf("input error???, bye\n");
        exit(1);
}
datasize = strtol( buffer, (char **)NULL, 0 );
printf("writeing %d bytes to discard\n", datasize );

printf("how big of packets? ");
fflush(stdout);
if( gets( buffer ) != buffer ) {
        printf("input error???, bye\n");
        exit(1);
}
pktsz = strtol( buffer, (char **)NULL, 0 );
printf("using %d (0x%x) byte packets\n", pktsz );

sz = pktsz;
i = datasize;
starttime = time( (long *)0 );
while( i > 0 ) {
        if( i < pktsz )
                sz = (int)i;
        if( write( s, buffer, sz ) != sz ) {
                perror("discard write failed");
                exit(1);
        }
        i -= sz;
}
endtime = time( (long *)0 );
```

```
        i =  endtime - starttime;
        if( i <= 0 ) {
                printf("wrote %d bytes in < 1 second\n", datasize );
                exit(0);
        }
        printf("wrote %d bytes in %d seconds\n", datasize, i );
        printf("   (%d Kbytes per second )\n",
            (datasize/(i))/1024 );
        exit(0);
}

chargen_stream( s )             /* TEST Character generator */
int s;
{
        long    datasize;
        time_t  starttime;
        time_t  endtime;
        char buffer[BUFSIZ];
        register long i;
        register int sz;

        printf("how many characters to get? ");
        fflush(stdout);
        if( gets( buffer ) != buffer ) {
                printf("input error???, bye\n");
                exit(1);
        }
        datasize = strtol( buffer, (char **)NULL, 0 );
        printf("receiving %d bytes from chargen\n", datasize );

        sz = BUFSIZ;
        i = datasize;
        starttime = time( (long *)0 );
        while( i > 0 ) {
                if( i < BUFSIZ )
                        sz = (int)i;
                if( (sz = read( s, buffer, sz )) < 0 ) {
```

```
                        perror("chargen stream read failed");
                        exit(1);
                }
                if( write( 1, buffer, sz ) != sz ) {
                        perror("write of chargen stream data to fd 1 failed");
                        exit(1);
                }
                i -= sz;
        }
        endtime = time( (long *)0 );

        i =  endtime - starttime;
        if( i <= 0 ) {
                printf("\ngot %d bytes in < 1 second\n", datasize );
                exit(0);
        }
        printf("\ngot %d bytes in %d seconds\n", datasize, i );
        printf("   (%d bytes per second )\n", datasize / i );
        exit(0);
}

chargen_dg( s )         /* Character generator */
int s;
{
        char buffer[BUFSIZ];
        register int sz;

        /* write to this guy to start him off (give him you address) */
        if( write( s, buffer, 2 ) < 0 ) {
                perror("chargen datagram startup write failed");
                exit(1);
        }
        if( (sz = read( s, buffer, BUFSIZ )) < 0 ) {
                perror("chargen datagram read failed");
                exit(1);
        }
        if( write( 1, buffer, sz ) != sz ) {
                perror("write of chargen datagram data to fd 1 failed");
```

```
		exit(1);
	}
	exit(0);
}

/*
 * Return a machine readable date and time, in the form of the
 * number of seconds since midnight, Jan 1, 1900.  Since gettimeofday
 * returns the number of seconds since midnight, Jan 1, 1970,
 * we must add 2208988800 seconds to this figure.
 */

machtime_stream( s )
int s;
{
	long	rawtime;

	if( read(s, &rawtime, 4 ) != 4 ) {
		perror("machtime stream failed");
		exit(1);
	}
	printf( "raw time is %d (0x%x)\n", rawtime, rawtime );
	rawtime -= 2208988800;  /* convert to unix time */
	printf("converted Unix time is %d\n", rawtime );
	printf("which translate to %s\n", asctime( localtime( &rawtime ) ) );
	exit(0);
}

machtime_dg( s )
int s;
{
	long	rawtime[512];

	/* prime the datagram socket - to send it your address */
	if( write(s, rawtime, 2 ) <= 0 ) {
		perror("machtime datagram write failed");
		exit(1);
	}
```

```
        if( read(s, rawtime, sizeof(rawtime) ) <= 0 ) {
                perror("machtime datagram failed");
                exit(1);
        }
        printf( "raw time is %d (0x%x)\n", rawtime[0], rawtime[0] );
        rawtime[0] -= 2208988800;       /* convert to unix time */
        printf("converted Unix time is %d\n", rawtime[0] );
        printf("which translate to %s\n", asctime( localtime( &rawtime[0] ) ) );
        exit(0);
}

daytime_stream( s )             /* Return human-readable time of day */
int s;
{
        char buffer[256];

        bzero( buffer, sizeof(buffer) );
        if( read(s, buffer, sizeof(buffer) - 1 ) <= 0 ) {
                perror("daytime stream failed");
                exit(1);
        }
        printf("THE TIME IS: %s\n", buffer );
        exit(0);
}

daytime_dg( s )         /* Return human-readable time of day */
int s;
{
        char buffer[256];
        struct sockaddr_in sin;
        int size;

        /* prime the datagram socket - to send it your address */
        if( write(s, buffer, 2 ) <= 0 ) {
                perror("daytime datagram write failed");
                exit(1);
        }
        bzero( buffer, sizeof(buffer) );
```

```
        size = sizeof(sin);
        if (read(s, buffer, sizeof(buffer) ) <= 0) {
                perror("daytime datagram failed");
                exit(1);
        }
        printf("THE TIME IS: %s\n", buffer );
        exit(0);
}

/* print usage message */
usage( s )
char *s;
{
        register struct biltin *bi;
        register char *proto;

        fprintf( stderr, "%s: %s \n", myname, s );
        fprintf( stderr, "    usage: %s [host] service proto\n", myname );
        fprintf( stderr, "      service proto is: \n" );
        for (bi = biltins; bi->bi_service; bi++) {
                if( bi->bi_socktype == SOCK_DGRAM )
                        proto = "udp";
                else
                if( bi->bi_socktype == SOCK_STREAM )
                        proto = "tcp";
                else {
                        proto = "????";
                }
                fprintf( stderr, "        %s %s\n", bi->bi_service, proto );
        }
        fflush( stderr );
        exit(1);
}
```

Connection establishment

This example initiates a server process for **ibmnews**, which has the following entry in the **/etc/services** file: `ibmnews   1524/tcp`

```
# include <stdio.h>
# include <errno.h>
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <netdb.h>

main(argn,argl)
int argn;
char **argl;
{
    int sockfd;          /* daemon socket file descriptor */
    int newsock;
    int on = 1;
    int debug = 0;
    struct sockaddr_in src, dst;
    struct servent *sp;
/*
 *  Note the linger structure for turning off (on) linger on tcp socket.
 */
    struct linger optval;

/*
 ** socket creation
 */
    if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
       error processing
    }
/*
 ** set our options before the bind.
 */
    if (debug) {
```

```
      if (setsockopt(sockfd, SOL_SOCKET, SO_DEBUG, (char *)&on, sizeof on)) {
         error processing
      }
   }

   optval.l_onoff = 0;
   optval.l_linger = 0;
   if (setsockopt(sockfd, SOL_SOCKET, SO_LINGER, &optval, sizeof (optval))) {
       error processing
   }
   if (setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, (char *)&on, sizeof on)) {
       error processing
   }
   if (setsockopt(sockfd, SOL_SOCKET, SO_KEEPALIVE, (char *)&on, sizeof on)) {
       error processing
   }
/*
 ** set up the address for server.
 */
   sp = getservbyname("ibmnews", "tcp");
   if (sp == NULL) {
       error processing
   }

   src.sin_family      = AF_INET;
   src.sin_addr.s_addr = INADDR_ANY;
   src.sin_port        = sp->s_port;

/*
 ** Ready for the bind.
 */

   if ( bind(sockfd, (char *) &src, sizeof(src)) < 0 ) {
       error processing
   }
```

```
/*
 ** Ready for listen.
 */

    if (listen(sockfd,5) < 0 ) {
        error processing
    }
/*
 ** Begin the service
 */

    while(1) {
        int fromlen;
        struct sockaddr_in fromend;

/*
 * Wait for a connection.
 */

        do {
            errno = 0;
            fromlen = sizeof fromend;
            newsock = accept(sockfd, &fromend, &fromlen);
        } while (newsock < 0 && errno == EINTR);

        if (newsock < 0) {
            error processing
            sleep(10);
            continue;
        }

/*
 * Ready to fork and provide service on newsock
 */
    }
}
```

Connectionless sockets

This example constructs a broadcast socket and initiates a broadcast. Note the use of the library routines to construct addresses and the use of the **getsockopt** subroutine to retrieve socket options. Port 1524 has the same **/etc/services** entry as in the previous example.

```
# include <stdio.h>
# include <errno.h>
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <arpa/inet.h>
# include <netdb.h>
#define PORT 1525
#define CAST 32

struct hostent *gethostbyname();
struct in_addr inet_makeaddr();

main(argn, argl)
int argn;
char **argl;
{

    char buf[256];
        int source;
        int test;
        int size;
        int net;
        int on = 1;
        int i;
    char buffer[BUFSIZ];
    unsigned char *ptr;
    struct sockaddr_in src, dst;
    struct hostent *hp;
    struct in_addr ina;
    struct sockaddr name;
```

```
/*
 ** use any available local port
 */

    src.sin_family      = AF_INET;
    src.sin_addr.s_addr = INADDR_ANY;
    src.sin_port        = 0;

/*
 * Socket creation
 */

    if ( (source = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        error processing
    }

/*
 * Set our options
 */

    if (setsockopt(source,SOL_SOCKET, SO_BROADCAST, (char *)&on, sizeof on)) {
        error processing
    }

/*
 * Check our options
 */

    size = sizeof(test);
    if (getsockopt(source,SOL_SOCKET, SO_BROADCAST,(char *)&test, &size)) {
        error processing
    }

    if (test != CAST) {
        error processing
    }
```

```
/*
 * Ready for the bind.
 */
    if ( bind(source, &src, sizeof(src)) < 0 ) {
        error processing
    }

/*
 ** get our hostname
 */

    if ( gethostname(buffer, sizeof buffer) ) {
        error processing
    }

/*
 ** get our address & official hostname
 */

    if ( (hp = gethostbyname(buffer)) == NULL ) {
        error processing
    }
/*
 ** extract the network part of address
 */

    ina = *(struct in_addr *)hp->h_addr;
    net = inet_netof(ina);

    dst.sin_port   = PORT;
    dst.sin_family = AF_INET;
/*
 ** Construct the broadcast address
 */
    dst.sin_addr = inet_makeaddr(net, INADDR_BROADCAST);
    ptr = (unsigned char *) &(dst.sin_addr);
    printf("ADDR %d:%d:%d:%d\n", *ptr, *(ptr+1), *(ptr+2), *(ptr+3));
```

```
/*
 ** The message
 */

    strcpy(buf, "test broadcast");

    for (i=0; i < 10; i++)
        if ( sendto(source, buf, strlen(buf), 0, &dst, sizeof(dst)) < 0 ) {
            error processing
        }
}
```

Input/Output Multiplexing

This example initiates a pair of sockets in the **UNIX** domain. Reads and writes are then made using the **select** system call.

```
# include <stdio.h>
# include <errno.h>
# include <sys/time.h>
# include <sys/types.h>
# include <sys/socket.h>
# include <netdb.h>
# define BSD_INCLUDES

main(argn,argl)
int argn ;
char **argl ;
{

    int sf[2];
    int fdnum;
    int i;
    char buffer[BUFSIZ];
    char *transmit = buffer;
    struct timeval wait;
    fd_set ones, twos;
    transmit = "Hello";
/*
 * Set timeouts for select.
 */
    wait.tv_sec = 1;
    wait.tv_usec = 0;

/*
 * Clear the set masks;
 */

    FD_ZERO(&ones);
    FD_ZERO(&twos);
```

```
/*
 * Open the sockets. File descriptors will be in array sf.
 */

    if (socketpair(AF_UNIX, SOCK_STREAM, 0, sf) !=0 ) {
        error processing
    }

/*
 * Start the write on sf[0]. Start the read on sf[1].
 */
    FD_SET(sf[0],&twos);
    FD_SET(sf[1],&ones);
    for (i=0; i < 10; i++) {
        fdnum = select(32,(fd_set *) 0, &twos,(fd_set *) 0, &wait);
        if (fdnum < 0) {
            error processing
        }
        if (!FD_ISSET(sf[0],&twos)) {
            error processing
        }
        if (write(sf[0],transmit,BUFSIZ) != BUFSIZ) {
            error processing
        }
        fdnum = select(FD_SETSIZE,&ones, (fd_set *) 0, (fd_set *) 0, &wait);
        if (fdnum < 0) {
            error processing
        }
        if (!FD_ISSET(sf[1],&ones)) {
            error processing
        }
        if (read(sf[1],transmit,BUFSIZ) != BUFSIZ) {
            error processing
        }
        else
            printf("received transmitted %s for %d time\n",transmit,i+1);
    }
```

```
/*
 * Switch
 */

    FD_CLR(sf[0],&twos);
    FD_CLR(sf[1],&ones);
    FD_SET(sf[0],&ones);
    FD_SET(sf[1],&twos);
    for (i=0; i < 10; i++) {
        fdnum = select(32,(fd_set *) 0, &twos,(fd_set *) 0, &wait);
        if (fdnum < 0) {
            error processing
        }
        if (!FD_ISSET(sf[1],&twos)) {
            error processing
        }
        if (write(sf[1],transmit,BUFSIZ) != BUFSIZ) {
            error processing
        }
        fdnum = select(FD_SETSIZE,&ones, (fd_set *) 0, (fd_set *) 0, &wait);
        if (fdnum < 0) {
            error processing
        }
        if (!FD_ISSET(sf[0],&ones)) {
            error processing
        }
        if (read(sf[0],transmit,BUFSIZ) != BUFSIZ) {
            error processing
        }
        else
            printf("received transmitted %s for %d time\n",transmit,i+1);
    }
}
```

Appendix C. Protocol Library Routines

About This Appendix

This appendix describes the **tcpm**, **tcp**, and **udp** libraries. These library routines are provided only for compatibility with existing programs. New programs should use the sockets interface to the protocols, as described in *AIX Operating System Technical Reference*. For examples of programs written to the sockets API, see Appendix B, "Examples" on page B-1.

Note: In Version 2.2.1 of the Interface Program, the **netioctl.h** include file has been removed. Existing programs that use this file must be rewritten.

The following **include** files are in the **/usr/include** directory:

in_extern.h
in_params.h
ip.h
notice.h
udp.h
task.h
taskm.h
q.h

Comment lines in these files explain their functions.

Note: In this appendix, the term ***socket*** or ***foreign socket*** refers to the 16-bit port number (not to the IP socket, which is the port number concatenated with the IP address).

tcpm

Purpose

Provides the layer that supports multiple TCP connections.

Library

/usr/lib/libtcpm.a

Syntax

#include <ip.h>
#include <taskm.h>

Description

The **tcpm** library provides support for multiple **tcp** connections, up to a maximum of 128 per process.

The **tcpm** library provides some support for the internal tasking system within a single AIX process. Task calls must be issued by the user. For more information about the tasking system, see "Interface Program Tasking System" on page C-10.

The routines in the **/usr/lib/libtcpm.a** library interface with the sockets library routines, which are described in *AIX Operating System Technical Reference*. When compiling a program that uses **libtcpm.a**, specify the **-ltcpm** option.

The **libtcpm.a** library contains the following routines.

Routines

int tcp_init (*stacksize***)**
  int *stacksize***;**

Initializes TCP and the tasking system, and sends and receives tasks. *stacksize* is the size of the stack of the current task. **tcp_init** returns **TRUE** if initialization succeeds, **FALSE** if it fails.

int tcp_open (*fh, lh, fs, ls, win, us_procs*)

```
long   fh;          /* foreign host to open to            */
long   lh;          /* local host to open to              */
short  fs;          /* foreign socket to open to          */
short  ls;          /* local socket to open to            */
int    win;         /* window size, see below             */
int    (*us_procs[])( );/* 6 user defined procedures, see below */
```

Immediately tries to open a connection to foreign host (*fh*) on foreign socket (*fs*) from the local socket (*ls*). If the local socket is 0, **tcp_open** selects an unused socket number greater than 1000. If the value for local host is 0, **tcp_open** selects the optimal interface from the ones that are configured.

Note: This routine returns immediately. To determine whether the connection is open, your current task must block/yield until the **us_open** routine completes. (**us_open** is described under "User-Defined Routines" on page C-8.)

A small, non-negative number (the connection ID) is returned if the routine opens an Internet connection with the specified hosts and sockets. If the connection is not opened, or if no more connection blocks are available, the routine returns a value of -1.

int tcp_passive_open (*fh, fs, ls, win, us_procs*)

```
long   fh;          /* foreign host to open to            */
short  fs;          /* foreign socket to open to          */
short  ls;          /* local socket to open to            */
int    win;         /* window size, see below             */
int    (*us_procs[])( );/* 6 user defined procedures, see below */
```

Performs a passive open on the specified foreign host, foreign socket, and local socket; that is, the process will accept incoming connection requests rather than attempting to initiate a connection. Both the foreign host and foreign socket may be 0.

Note: This routine returns immediately. To determine whether the connection is open, your current task must block/yield until the **us_open** routine completes. (**us_open** is described under "User-Defined Routines" on page C-8.)

A small, non-negative number (the connection ID) is returned if the routine opens an Internet connection with the specified hosts and sockets. If the connection is not opened, or if no more connection blocks are available, the routine returns a value of -1.

```
int tcp_listen(ls, win, us_procs)
    short  ls;             /* local host to open to          */
    int    win;            /* window size, see below          */
    int    (*us_procs[])( );/* 6 user defined procedures, see below  */
```

Listens for a server connection on the specified local socket with the foreign host and port unspecified. A small, non-negative number (the connection ID) is returned if the routine opens an Internet connection with the specified hosts and sockets. If the connection is not opened, or if no more connection blocks are available, the routine returns a value of -1.

A child process runs the connection while the parent process returns to listen on the connection. A double fork makes the child process a child process of **/etc/init**; thus, clean up can be done after the process exits without the parent process having to wait for the exit. To determine whether the connection is open, the child process must block/yield until the **us_open** routine completes; **us_open** is described under "User-Defined Routines" on page C-8.

```
int tcp_puts (con, str)
        int    con;        /* connection id                 */
        char   *str;        /* string to output              */
```

Functions similar to the AIX **puts** routine except that the network connection must be specified. This routine returns **FALSE** if there is not enough room for the entire string in the TCP send buffer; the task that issues this call should either block or yield until room in the TCP send buffer becomes available.

Note: This routine does not awaken the send task (to send the character) unless the TCP send buffer is full. To force the character to be sent, use the **tcp_flush** routine.

```
int tcp_putc (con, c)
        int    con;        /* connection id                 */
        char   c;          /* character to output            */
```

Tries to write the character to the network. This routine returns **FALSE** if the TCP send buffer is full, **TRUE** otherwise.

Note: This routine does not awaken the send task (to send the character) unless the TCP send buffer is full. To force the character to be sent, use the **tcp_flush** routine.

```
int tcp_write (con, buf, len)
      int    con;      /* connection id         */
      char   *buf;     /* buffer to output      */
      int    len;      /* length of buffer      */
```

Functions similar to the AIX **write** routine except that the network connection must be specified. This routine returns **FALSE** if there is not enough room for the entire string in the TCP send buffer.

Note: This routine does not awaken the send task (to send the character) unless the TCP send buffer is full. To force the character to be sent, use the **tcp_flush** routine.

```
int tcp_close (con)
      int    con;      /* connection id         */
```

Closes the local side of the current connection by sending **FIN** to the foreign host. This routine returns immediately, but the user must wait for the user-defined procedure **us_close** to be called to indicate that the connection is completely closed.

```
int tcp_urgent (con)
      int    con;      /* connection id         */
```

Indicates that all data in the TCP send buffer is urgent. Subsequent **tcp_urgent** calls can be made, which extend the length of the urgent data portion.

```
int tcp_clean (con)
      int    con;      /* connection id         */
```

Erases the connection. This routine can eliminate listening TCP connections or active TCP connections when the foreign host is down or does not respond.

```
int tcp_abort (con)
      int    con;      /* connection id         */
```

Sends **TCP RESET** to the foreign host and calls the procedure **us_close** with the value of *reason* set to **FALSE**. This routine returns immediately, so the user must wait until *us_close* is called before the reset is complete.

```
int tcp_flush (con)
      int    con;      /* connection id         */
```

Tries to send all outstanding data in the TCP send buffer to the foreign host.

```
int tcp_nput (con)
    int con;            /* connection id                */
```

Returns the amount of free space in the TCP send buffer. This number can be used later to determine how much data a user can put in the TCP send buffer.

```
int tcp_setu (con, ucb)
    int   con;          /* connection id                */
    char  *ucb;         /* user control block           */

char *tcp_getu (con)
    int   con;          /* connection id                */
```

Two routines that make it possible to set and access a user-defined character pointer field for the connection. The value given to this field can be a pointer to the task that will send data through the network. The pointer is supplied to **tk_wake** when the **us_buf** call occurs.

```
tcp_new_window (con, window)
    int   con;          /* connection id */
    int   window;       /* window size   */
```

Changes the value of the local window. This routine should be used primarily to open a local window that has shrunk to 0; it is the only way to do so. If this routine is used at other times, poor performance and confusion from the foreign host may result.

```
conn_info (con, plhost, pfhost, plsock, pfsock)
    int   con;                  /* connection id        */
    long  *plhost, *pfhost;     /* local host, foreign host /*
    short *plsock, *pfsock;     /* local socket, foreign socket /*
```

Gets complete information about the connection: local host, foreign host, local port, and foreign port. If the user does not need information about a particular argument (for example, foreign host), he should pass **NULL** as the argument.

```
tcp_debug (con, dbg)
     int    con;        /* connection id         */
     int    dbg;        /* level of debug, valid are 0, 1, 2;
                                   are additive */
extern char    *logfname;    /* log file name for trace output
                        0 = trace messages
                        1 = error messages
                        2 = shows received packets
                        3 = shows sent packets        */
extern int     TCPDEBUG;
```

Turns on or off different levels of data tracing. In order to use this routine, the external variable *logfname* should be set to the name of the file into which all tracing will be written, and **TCPDEBUG** should be set to 1 before the call is made to **tcp_init**. **TCPDEBUG** controls default tracing for all TCP connections. This default may be changed by using the **tcp_debug** routine.

To look at TCP headers, use the **setsockopt** subroutine and the **trpt** command. See *AIX Operating System Technical Reference* for more information on **setsockopt** and "**trpt**" on page 2-89 for more information on the **trpt** command.

tcp_ioff (*conn*);   /* connection id */

Turns interrupts off; used while writing data to the display.

tcp_ion (*conn*);   /* connection id */

Turns network interrupts on.

tm_on ( );

Wakes up the timer task to process any events that went off while timer interrupts were disabled, and to start a new alarm.

tm_off ( );

Turns off timer interrupts; useful when writing data to the display.

tk_block ( );

Blocks the current task and forces it to be rescheduled.

char *tk_cur;

Points to the task control block of the currently running task; a global variable.

tk_yield ( );

Yields the processor to any other task that can be run.

```
tk_wake (tk)
task *tk;              /* task control block              */
```

Wakes up a specified task. The *tk* parameter is a pointer to a control block, which is defined in the **taskm.h** header file.

```
tk_setef (tk, ef)
task *tk;              /* task control block              */
task ushort ef;           /* event flag                   */
```

Wakes up a task and sets a specified event flag for it. The *tk* parameter is a pointer to a control block, which is defined in the **taskm.h** header file. The *ef* parameter specifies the event flag.

User-Defined Routines

Following are definitions of the six user-defined routines in the array that TCP calls. The names of these procedures are not fixed; that is, they can be renamed to meet your requirements.

User-defined routines must be in the following order in the array definition:

```
us_procs    [0]  =  us_open
            [1]  =  us_dispose
            [2]  =  us_close
            [3]  =  us_forclose
            [4]  =  us_timeout
            [5]  =  us_buff
```

```
us_open (con, fh, fs)
    int    con;         /* connection id            */
    long   fh;          /* foreign host             */
    short  fs;          /* foreign socket           */
```

Called when the TCP connection reaches the **ESTABLISHED** state (that is, when both the local and foreign hosts can send data). Most applications should use this routine to wake up a task that writes data to the network.

```
int us_dispose (con, ptr, len, urg)
    int   con;      /* connection id                  */
    char  *ptr;     /* pointer to buffer to dispose   */
    int   len;      /* length of buffer               */
    int   urg;      /* urgent pointer (see below)     */
```

Called to send data to the user's program. The *len* parameter specifies the length of the urgent data in the buffer. A non-negative value for *urg* represents an offset in the buffer to the urgent data. If no urgent data is present, the value of *urg* is -1.

```
us_close (con, reason)
    int   con;      /* connection id                  */
    int   reason;   /* 1 = normal close; 0 = abnormal close */
```

Called to indicate that both sides have closed and that the connection block is erased. The value of *reason* is **TRUE** if the connection closes normally, **FALSE** if the connection closes abnormally due to a **TCP RESET**.

```
us_forclose (con, reason)
    int   con;      /* connection id                  */
    int   reason;   /* 1 = normal close; 0 = abnormal close */
```

Called if the foreign host sends **FIN** (that is, the foreign host closes its side of the connection) or **TCP RESET**. The **us_forclose** (foreign close) routine immediately closes the local side of the connection. The value of *reason* is **TRUE** if the connection closes normally, **FALSE** if the connection closes abnormally due to a **TCP RESET**.

```
us_timeout (con)
    int   con;      /* connection id                  */
```

Called when a timeout occurs.

```
us_buff (con, count)
    int   con;      /* connection id                  */
    int   count;    /* amount of free space in TCP send buffer */
```

Called when the output buffer is filled and buffer space is not available. Usually a task that writes to the network should block itself when it determines no more space is available in the TCP send buffer. The main purpose of **us_buff** is to awaken the blocked task, which it always should do.

Interface Program Tasking System

A ***task*** is a process together with the procedures that run in the process. The procedures are often from different protocol layers. ***Tasking*** is a way of organizing procedures and processes to form layers that:

- Preserve layer modularity
- Process the asynchronous data of layers efficiently.

One task cannot preempt another, but tasks can be preempted by routines (which are signal driven). A task stops running when it calls a blocking routine in the tasking module.

Tasks that are internal to the same protocol layer are said to share ***states***. In task programming, one task can awaken another only if the two tasks share states. For example, if one task is to awaken another, it must have access to the task control block of the other task.

The tasking system is used by TCP and applications that use TCP. Each task has a stack and a task control block. A task control block is allocated by calling the **calloc** structure. The pointer that **tk_fork** returns is a pointer to this structure.

Figure C-1 on page C-11 compares the memory organization of a single AIX process running on the RT system without tasking to the memory organization of a single AIX process running with tasking. In the lower horizontal row (the one that represents memory organization with tasking), **TS** refers to a task stack and **TCB** refers to a task control block.

Process without Tasking →	3FFFFFFF	Stack	2FFFFFFF	Static Data	1FFFFFFF	Program Code
Process with Tasking →	3FFFFFFF	TS TS TS	2FFFFFFF	TCB TCB TCB	1FFFFFFF	Program Code

Figure C-1. Relationship of Tasking to Memory Organization

These memory organizations are similar except that, for the process running with tasking, the stack is partitioned for multiple tasks and the static data area contains task control blocks. Each task stack is organized like the original program stack.

To use the tasking system, include the **taskm.h** header file. **taskm.h** associates a task control block with every task; the task control block is a structure that provides a new task type. This include file also provides ***events***, or binary semaphores, that allow communication between tasks and allow tasks to determine why they are started. The task control block is included in **taskm.h**.

A pointer to a task control block is analogous to a process ID. Task control blocks are chained together by the **tk_elt** member of the task control block structure to form a circular list. The tasking scheduler is a non-preemptive, round-robin scheduler that simply loops through the circular list of tasks until it finds one that can be run (that is, a task with a **tk_evf** that is set **TK_TRUE**) and then switches to that task. A combination of the **setjmp** and **longjmp** routines performs the context switch. The global variable **tk_cur** is a pointer to the task control block of the task that is currently running.

Generally, task wakeups should be treated as hints. When a task runs, it should try to determine why it was started and perform accordingly. Part of the function of a task should be to respond appropriately to nonspecific wakeups. Tasks are forked as runnable; therefore, the following structure is correct for a task:

```
task_definition(tk)
struct task    *tk;
{
/* all local declarations       */

        set_task(tk);    /* must be first executable statement */

        for (;;) {
                if (no_work) {
                        tk_block();
                        continue;
                        }
                do_the_right_thing;
                }
}
```

tcp

Purpose

Provides the Transmission Control Protocol (TCP) layer.

Library

/usr/lib/libtcp.a

Syntax

#include <ip.h>
#include <task.h>

Description

The TCP library provides a set of routines suitable for reliable data transmission.

The TCP library provides some support for the internal tasking system within a single AIX process. Task calls must be issued by the user.

The routines in the **/usr/lib/libtcp.a** library interface with the sockets library routines, which are described in *AIX Operating System Technical Reference.* When compiling a program that uses **libtcp.a**, specify the **-ltcp** option.

The **libtcp.a** library contains the following routines:

tcp_init (*stksiz, 0, ofcn, infcn, cfcn, fcfcn, tmofcn, sfcn***)**
int *stksiz*;
int *extsiz*;
int **(****ofcn***) ()**;
int **(****infcn***) (***prt, len, urgent***)**;
int **(****cfcn***) (TRUE|FALSE)**;
int **(****fcfcn***) ()**;
int **(****tmofcn***) ()**;
int **(****sfcn***) ()**;

Initializes the TCP layer. This routine starts the internal tasking system and sets the pointers to the user routines that can be called. It does not attempt to open the connection. That function is performed by **tcp_open**. When it returns, the caller is running as the first task on the primary process stack. **tcp_init** sets the following pointers:

stksiz Value for the depth that the thread or task will require.

ofcn Called once TCP reaches the ESTABLISHED state (both the local and foreign hosts are able to send data). Most commands use this routine to awaken a task that writes data to the network.

infcn Called to send some data to the user program. TCP does not acknowledge data until this routine returns. *infcn* sets the following pointers:

char * prt Sets pointer to the beginning of the data.

int len Returns the number of bytes of data sent.

ushort urgent The normal value is -1. If the value is greater than 0, it implies urgent data in the buffer and points to it.

cfcn Called to signal that both sides have closed, and the connection block is erased. The returned status is **TRUE** if the connection closed normally, or **FALSE** if the connection closed abnormally due to a TCP RESET.

fcfcn Called if the foreign host sends a FIN (the foreign host has closed its half of the connection) before a close in the local host. This initiates an immediate close in the local host.

tmofcn Called when a timeout has occurred.

sfcn Called when the previously full output buffer has buffer space available. TCP immediately blocks the writing process, which prevents writing to a connection before the connection is open. Also, if TCP determines the foreign window is full, or when the current output packet is full, any attempt to write to the network blocks the writing process. In these

three cases, this call signals that the cause of the block has disappeared. The user routine should always awaken those tasks that are blocked.

tcp_open (*fh, fs, win*)
in_name *fh*;
ushort *fs*;
int *win*;

Opens an active **tcp** connection to foreign host *fh*, on foreign socket *fs*, with a receive window size of *win* bytes. Gets a unique local socket on which to open the connection. Returns **FALSE** if unable to open an Internet connection with the specified hosts and sockets. Otherwise it returns the local socket on which the connection is opened. This routine forks a child process to handle the connection; it does not wait until the connection is actually opened before it returns. Instead, the *ofcn* (user open routine) specified in the call to **tcp_init** is called when the connection is successfully opened. (The **tcp_init** call must precede the **tcp_open** call.) **open** allows a single AIX process on a single port; the tasking system allows multiple threads within each process.

tcp_close ()

Initiates the TCP closing sequence. This routine returns immediately. When the close is complete, the *cfcn* (user close routine) is called.

tcp_lstn (*ls, win*)
ushort *ls*;
int *win*;

Listens for a server connection on the specified local socket with the foreign host and port unspecified. Returns **FALSE** if unable to open an Internet connection with the specified hosts and sockets, or **TRUE** otherwise. This routine returns immediately. It does not wait until the connection is actually opened before returning. When a connection is successfully opened, a process performs a **fork** call while in the **tcp_rcv** routine, the child process proceeds to open a specified Internet connection with the appropriate parameters, and the *ofcn* (user open routine) is in the child process. The parent returns to listen on the connection. **lstn** supports one TCP process on one or more ports; the tasking system allows multiple threads within each process.

tc_put (*c*)
char *c*;

Inserts a character into the send buffer for transmission, but does not wake up the TCP sending task. It assumes that more data immediately follows. It returns **TRUE** if there is more room to store data in the buffer. Otherwise, it returns **FALSE**.

tc_fput (*c*)
char *c*;

Inserts a character to be transmitted into the send buffer, and wakes up the TCP sending task to send it. It returns **TRUE** if there is more space in the buffer to store data. Otherwise, it returns **FALSE**.

tcpurgent ()

Indicates that urgent data is present. Sets the urgent pointer to the current data length and awakens TCP to send it.

tcp_passive_open (*fh, fs, ls, win*)
in_name *fh*;
ushort *fs*;
ushort *ls*;
int *win*;

Indicates that the process will accept incoming connection requests rather than attempt to initiate a connection. Often the process requesting a passive OPEN will accept a connection request from any caller. In this case, a foreign socket of all 0's is used to denote an unspecified socket. Unspecified foreign sockets are allowed only on passive OPENs. In this case, *fh* specifies the foreign host, *fs* specifies the foreign socket, *ls* specifies the local socket, *win* specifies the receive window size. **passive_open** allows a single AIX process on a single port; the tasking system allows multiple threads within each process.

tcp_debug (*onoff*)
int *onoff*;

Turns TCP debugging (data-level tracing) on or off.

For packet-level debugging, use the **setsockopt** subroutine and the **trpt** command. See *AIX Operating System Technical Reference* for more information on **setsockopt** and "**trpt**" on page 2-89 for more information on the **trpt** command.

tcp_abort ()

Makes a user-level request to abort connection.

tcp_flush ()

Wakes up a TCP send task so that all outstanding data will be sent over the TCP connection.

tc_write (*buf, count*)
char **buf*;
int *count*;

Writes a block of data to a TCP connection. *buf* is a counter and *count* is an integer.

tm_on ( )

Wakes up the timer task to process any events that went off while timer interrupts were disabled, and to start a new alarm.

tm_off ( )

Turns off timer interrupts; useful when writing data to the display.

tk_block ( )

Blocks the current task and forces it to be rescheduled.

tk_yield ( )

Yields the processor to any other task that can be run.

tk_wake (*tk*)
task **tk*;

Wakes up a specified task. The *tk* parameter is a pointer to a control block, which is defined in **task.h**.

tk_setef (*tk*, *ef*)
task **tk*;
task ushort *ef*;

Wakes up a task and sets a specified event flag for it. The *tk* parameter is a pointer to a control block, which is defined in **task.h**. The *ef* parameter specifies the event flag.

udp

Purpose

Provides the User Datagram Protocol (UDP) layer.

Libraries

/usr/lib/libU.a
/usr/lib/libudp.a

Syntax

#include <in_extern.h>
#include <udp.h>

Description

The UDP provides a set of routines that allow commands to send messages (datagrams) to and receive messages from the other processes, possibly running on other hosts attached to other networks. It places no restriction on the format of data in the messages. The maximum length of a message is 512 bytes in the AIX implementation. The protocol is transaction-oriented. Reliability and duplicate protection of datagrams are not guaranteed.

The routines in the **/usr/lib/libudp.a** and **/usr/lib/libU.a** libraries provide a user interface to the Internet protocols supported by AIX. (If you perform data tracing, link to **libudp.a**; otherwise, link to **libU.a**.) The **udp** library provides a basic set of routines to allocate, free, send, and receive UDP packets.

In order to use these routines, the user program must include a header file, containing type and structure definitions, and so on. To use the **udp** library, the **udp.h** and **in_extern.h** header files must be included. The program must then be linked using the **-l udp** flag to include the **udp** library. In addition, the program must include the standard I/O library to use this library. This means that the **-lS** flag must also be used when the program is linked.

Routines

The **/usr/lib/libudp.a** library contains the following routines:

udp_open(*fhost, fsock, lsock***)**
in_name *fhost*;
ushort *fsock*;
ushort *lsock*;

Opens a UDP connection to the specified foreign host with the specified local and foreign sockets. The foreign host address is a standard network address. The combination of foreign host, local socket, and foreign socket must be unique among all connections currently in progress on this host. The **udp_socket** routine may be used to obtain a unique local socket number. Note that any or all of the fields may be **NULL** to indicate **all**. The routine returns a file descriptor to be used in future read and write calls, or -1 for error. The external variable **errno** contains the error code.

ushort **udp_socket()**

Returns a unique 16-bit value for the local UDP socket. It calls the operating system to obtain a unique value. The socket returned is > 1000 (decimal) to ensure that it is a unique socket.

udp_close(*fd***)**
int *fd*;

Closes the specified UDP connection. The parameter is an AIX file descriptor returned by the **udp_open** call.

caddr_t **udp_alloc(***datalen, optlen***)**
int *datalen*;
int *optlen*;

Allocates the space for a UDP packet, with enough space for a data area of length *datalen* and enough space for *optlen* bytes of Internet options, plus room at the beginning for the local net, Internet, and UDP headers. It returns a pointer to the packet or **NULL** if unable to allocate. Also, it sets up the length fields of the Internet packet. These macros return pointers to the Internet header, the UDP header, and the UDP data portions of the packet, respectively. They are described under “Macros and Type Definitions” on page C-21.

udp_free(*pkt*)
caddr_t *pkt*;

Frees the packet pointed to by *pkt*. The packet must have been allocated by **udp_alloc**. It is an error to attempt to free something not previously allocated by **udp_alloc**. Also, the user is responsible for removing any remaining references to the packet.

udp_read(*fd*, *buf*, *len*)
int *fd*;
caddr_t *buf*;
int *len*;

Attempts to read the next available UDP packet on the specified connection into the specified buffer of length *len*. It returns the length of the received packet in bytes. If no packet is available, it returns 0. Note that this routine is non-blocking. It also validates the UDP length and checksum of the received packet. If either are invalid, the packet is dropped and 0 is returned. The **udp_data** macro can be used to obtain a pointer to the start of the data area of the packet.

udp_bread(*fd*, *buf*, *len*, *timeout*)
int *fd*;
caddr_t *buf*;
int *len*;
int *timeout*;

Performs a blocking read for UDP packets, with a timeout on the read. It waits *timeout* seconds for a UDP packet to arrive on the specified UDP connection, then reads it into the specified buffer. If a packet is presently available, the routine reads it and returns immediately. It returns the length of the received packet in bytes, or 0 if the timeout expires before a valid UDP packet is received. It also returns 0 on network error, or if interrupted by a signal. In these cases **errno** contains the error code.

udp_time ( )

Attempts to find the current time by requesting it from other UDP time servers. **udp_time** builds a time server request for each of the known time servers, then sends the request and waits for a response. If the request is answered, **udp_time** returns the time in seconds since midnight, 1-Jan-70 GMT. If the request is not answered (that is, no time server is running on the network), **udp_** returns a value of 0.

udp_write(*fd*, *buf*, *datalen***)**
udp_writex(*fd*, *buf*, *datalen*, *lh* **)**
int *fd*;
caddr_t *buf*;
int *datalen*;
int *lh*;

Writes the specified packet out to the specified net connection. The *buf* parameter is a pointer to a packet buffer as allocated by **udp_alloc**. The write routine assumes that the foreign address, Internet packet ID, protocol, and type of service fields (in the Internet header), and the local and foreign sockets (in the UDP header) have been supplied by the caller. If the Internet ID field is 0, the system assigns a unique ID. The system provides the rest of the UDP header, including checksum, then writes it to the net connection. The system returns the number of bytes written, if successful. If not successful, the system returns -1, and the external variable **errno** contains the error code. If the value of *lh* is 0, the optimal interface of those configured is selected.

long resolve_name (*name***)**
**register char ****name*;

Resolves a host name into an Internet address. Three name formats are accepted:

- A character string host name.
- An octal or decimal host number in the form *net, subnet, rsd, host*. Values for *net*, *subnet*, and *rsd* can be left blank or left out entirely; they default to the local net or subnet.
- A 32-bit hex number, preceded by a #, which is used without interpretation as the host number.

If a character string name is supplied, it is first looked up in a local host table. If it is not found there, the routine goes off to Internet name servers to try to resolve the name.

Macros and Type Definitions

In addition to the routines previously described, several data type definitions and macros are supplied in the **udp.h** header file to simplify writing network programs.

The following data types are defined:

ushort An unsigned 16-bit integer, used many places in the UDP header.

caddr_t An address in memory. On the RT system, this is a character pointer. Packet pointers and other pointers to untyped data are of this type.

The UDP level provides the following macros to obtain pointers to various useful portions of UDP packets and other useful data:

udp_head(*pip*) Returns a pointer (**struct udp ***) to the UDP packet when given a pointer to the start of the ***internet** header.

udp_data(*pup*) Returns a pointer (**caddr_t**) to the start of the data portion of a UDP packet, given a pointer (**struct udp * pup**) to the UDP header. The pointer is a memory address, and can be cast to the appropriate type.

Figures

Glossary

This glossary contains a list of some of the common terms that you may read or hear when working with data communications. The terms are described only as they relate to data communications.

access. The manner in which files or data sets are referred to by the computer.

adapter. See *communications adapter*.

address field. The part of a packet containing addressing information.

addressing. (1) The way that the sending or control station selects the station to which it is sending data. (2) A means of identifying storage locations.

American National Standard Code for Information Interchange (ASCII). The code developed by ANSI for information interchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters.

American National Standards Institute (ANSI). An organization sponsored by the Computer and Business Equipment Manufacturers Association for establishing voluntary industry standards.

bandwidth. The difference, in hertz, between the two limiting frequencies of a band.

baseband system. A system whereby information is encoded, modulated, and impressed on the transmission medium. At any point on the medium, only one information signal at a time is present.

bit rate. The speed at which serialized data is transmitted, usually expressed in bits per second.

block. A group of records that is recorded, processed, or sent as a unit.

bridge. In the connection of local loops, channels, or rings, the equipment and techniques used to match circuits and facilitate accurate data transmission.

broadband. Transmission media and techniques that use a broad frequency range, divided into sub-bands of narrower frequency.

cable. The physical media for transmitting signals; includes copper conductors and optical fibers.

cancel. To end a task before it is completed.

carrier. A continuous frequency that can be modulated with a second (information-carrying) signal.

carrier sense multiple access with collision detection (CSMA/CD). The generic term for a class of medium access procedures that (1) allows multiple stations to access the medium at will without explicit prior coordination, (2) avoids contention by way of carrier sense and deference, and (3) resolves contention by way of collision detection and transmission.

channel. A path along which data passes.

character. A letter, digit, or other symbol.

client. On a network, the computer requesting services or data from another computer.

clock. A device that generates periodic signals used for synchronization.

coaxial cable. A cable consisting of one conductor, usually a small copper tube or wire, within and insulated from another conductor of larger diameter, usually copper tubing or copper braid.

collision. A condition caused by multiple overlapping transmissions on the medium, which results in garbled data.

communication channel. An electrical path that facilitates transmission of information from one location to another.

communications. See *data communications.*

communications adapter. A hardware feature that enables a computer or device to become a part of a data communications network.

communications line. The line over which data communications takes place; for example, a telephone line.

configuration. The group of machines, devices, and programs that make up a data processing system.

confirmation. A transmission by a receiver that permits a sender to continue.

console. A part of a computer used for communications between the operator or maintenance engineer and the computer.

console display. A display that can be requested only at the system console. From a console display an operator can display, send, and reply to messages and use all control commands.

contention. A condition on a communications channel when two stations attempt to use the same channel simultaneously.

contention resolution. The process of resolving contention (medium access control conflicts) according to a defined algorithm.

control block. A storage area used by a program to hold control information.

control character. A character, occurring in a particular context, that initiates, modifies, or stops any operation that affects the recording, processing, transmission, or interpretation of data (such as carriage return, font change, and end of transmission).

CSMA/CD. See *carrier sense multiple access with collision detection (CSMA/CD).*

current host. See *local host.*

daemon. A background process that is usually started at system start, runs continuously, and performs a function required by other processes.

data circuit. Associated transmit and receive lines that provide a means of two-way data communications.

data communications. The transmission of data between a combination of computers and remote devices (usually over a long distance).

data link. The equipment and rules (protocols) used for sending and receiving data.

data stream. All information (data and control information) transmitted over a data link.

digital data. Data represented by on and off conditions called bits.

display station. A device that includes a keyboard from which an operator can send information to the system and a display screen on which an operator can see the information sent to or received from the computer.

distortion. An undesirable change in a data communications signal.

dotted decimal. A common notation for Internet host addresses, which divides the 32-bit address into four 8-bit fields. The value of each field is specified as a decimal number and the

fields are separated by periods (for example, 010.002.000.052, or 10.2.0.52).

echo. A reflected signal on a communications channel.

emulation. Imitation; for example, the imitation of a computer or device.

enable. In interactive communications, to load and start a subsystem.

foreign host. Any host on the network except the one at which a particular operator is working; sometimes called *remote host.*

gateway. An entity operating above the link layer, which translates, when required, the interface and protocol used by one network into those used by another distinct network.

host. (1) The primary or controlling computer in the communications network. (2) A computer attached to a network.

interface. A common boundary, but not of internal connections.

interrupt. To take an action at a receiving station that causes the sending station to end a transmission.

LAN. See *local area network.*

local. Pertaining to a device, file, or system that is accessed directly from your system, without the use of a communications line. Contrast with *remote.*

local area network. A network in which communications are limited to a moderate-sized geographic area (1 to 10 km) such as a single office building, warehouse, or campus. A local network depends upon a communications medium capable of moderate to high data rate, and normally operates with a consistently low error rate.

local host. The host on the network at which a particular operator is working; sometimes called *current host.*

medium (media). The material in or on which data may be represented (for example, twisted pairs, coaxial cables, and optical fibers).

network. A collection of data processing products connected by communication lines for information exchange between stations.

network adapter. Circuitry that allows devices using a directly attached network to communicate with the system.

network management. The conceptual control element of a data station that interfaces with all of the layers of that data station and is responsible for the resetting and setting of control parameters, obtaining reports of error conditions, and determining if the station should be connected to or disconnected from the medium.

node. See *host.*

packet. The data of one transaction between a host and its network. A packet usually contains a network header, followed by one or more headers used by high-level protocols, followed by data blocks.

physical layer (or level). The lowest layer of network design as specified by the ISO Open System Interconnection (OSI) reference model. This layer is responsible for interfacing with the medium, detecting and generating signals on the medium, and converting and processing signals received from the medium and from the data link layer.

port. (1) An access point for data entry or exit. (2) An entrance to or exit from a network.

protocol. Rules for transferring data.

process. A program in execution.

remote. Pertaining to a device, file, or system that is accessed by your system through a communications line. Contrast with *local.*

remote host. See *foreign host.*

retransmit. To repeat the transmission of a message or segment of a message.

retry. To resend a transmission that did not achieve the desired or intended result; usually follows a timeout.

route. A path defined for sending data across a network.

route table. A structure in memory that describes, for the computer, all of the routes that are currently defined.

server. On a network, the computer that contains the data to be accessed.

session. The logical connection by which a host program or device can communicate with a program or device at a remote location.

socket. (1) A unique host identifier created by the concatenation of a port identifier with an IP address. (2) A port identifier.

status. The state of affairs or the condition of a station that determines its ability to enter into exchanges of control or information.

telecommunications. Transmitting signals over long distance.

teleprocessing. Processing data that is received from or transmitted to a remote location via communication channels.

terminal. A device, usually equipped with a keyboard and a display device, capable of sending and receiving information over a communications line.

timeout. Measurement of time interval allotted for certain events to occur (such as a response to polling or other controls) before corrective (recovery) action is taken.

transfer. To send data to one place and to receive data at another place.

transmission control characters. Special characters that are included in a message to control communication over a data link. For example, the sending station and the receiving station use transmission control characters to exchange information; the receiving station uses transmission control characters to indicate errors in data it receives.

transparent. In communications, pertaining to transmissions that have no possibility of interference with data link control, regardless of format or content. Transparent transmissions are unrecognized by data link controls.

well-known host name. A conventional name associated with an IP address on a particular network (for example, **nameserver** and **timeserver**).

well-known port. A conventional port assignment used by hosts that support the same protocols, whether or not the hosts are on the same network.

wide area network. A network that provides data communication capability in geographic areas larger than those serviced by local area networks.

work station. A device that lets people transmit information to or receive information from a computer; for example, a display station or printer.

Index

D

P

R

S

T

U

V

W

X

IBM RT Interface Program for use with TCP/IP — **SC23-2005-1**

Book Title — **Order No.**

Book Evaluation Form

Your comments can help us produce better books. You may use this form to communicate your comments about this book, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Please take a few minutes to evaluate this book as soon as you become familiar with it. Circle Y (Yes) or N (No) for each question that applies and give us any information that may improve this book.

Y N Is the purpose of this book clear?

Y N Is the table of contents helpful?

Y N Is the index complete?

Y N Are the chapter titles and other headings meaningful?

Y N Is the information organized appropriately?

Y N Is the information accurate?

Y N Is the information complete?

Y N Is only necessary information included?

Y N Does the book refer you to the appropriate places for more information?

Y N Are terms defined clearly?

Y N Are terms used consistently?

Y N Are the abbreviations and acronyms understandable?

Y N Are the examples clear?

Y N Are examples provided where they are needed?

Y N Are the illustrations clear?

Y N Is the format of the book (shape, size, color) effective?

Other Comments

What could we do to make this book or the entire set of books for this system easier to use?

Optional Information

Your name ____________

Company name ____________

Street address ____________

City, State, ZIP ____________

No postage necessary if mailed in the U.S.A.

Tape

Please Do Not Staple

Tape

Cut or Fold Along Line

Fold and Tape

Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758-3493

The IBM RT system
Family

Reader's Comment Form

IBM RT Interface Program for use with TCP/IP

SC23-2005-1

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

For prompt resolution to questions regarding set up, operation, program support, and new program literature, contact your IBM representative, your IBM Authorized Dealer, or your IBM Authorized Remarketer.

Comments:

Tape

Please Do Not Staple

Tape

Cut or Fold Along Line

Fold and Tape

Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758-3493

The IBM RT system
Family

Reader's Comment Form

IBM RT Interface Program for use with TCP/IP

SC23-2005-1

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

For prompt resolution to questions regarding set up, operation, program support, and new program literature, contact your IBM representative, your IBM Authorized Dealer, or your IBM Authorized Remarketer.

Comments:

Tape

Please Do Not Staple

Tape

Cut or Fold Along Line

Fold and Tape

Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758-3493

IBM

The IBM RT system
Family

Reader's Comment Form

IBM RT Interface Program for use with TCP/IP

SC23-2005-1

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

For prompt resolution to questions regarding set up, operation, program support, and new program literature, contact your IBM representative, your IBM Authorized Dealer, or your IBM Authorized Remarketer.

Comments:

Tape

Please Do Not Staple

Tape

Cut or Fold Along Line

Fold and Tape

Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758-3493

IBM

The IBM RT system
Family

Reader's Comment Form

IBM RT Interface Program for use with TCP/IP

SC23-2005-1

Your comments assist us in improving our products. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

For prompt resolution to questions regarding set up, operation, program support, and new program literature, contact your IBM representative, your IBM Authorized Dealer, or your IBM Authorized Remarketer.

Comments:

Tape

Please Do Not Staple

Tape

Cut or Fold Along Line

Fold and Tape

Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 997, Building 998
11400 Burnet Rd.
Austin, Texas 78758-3493

© IBM Corp. 1985, 1987, 1988
All rights reserved.

International Business
Machines Corporation
11400 Burnet Road
Austin, Texas 78758

Printed in the
United States of America

SC23-2005-1

IBM®

SC23-2005-1